MASTER EN SCIENCES INFORMATIQUES

Aide à l'édition de spécifications orientées agents de systèmes de production manufacturiers

Baudoin, Philippe

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 2004-2005

Aide à l'édition de spécifications orientées agents de systèmes de production manufacturiers

Philippe Baudoin

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique

Résumé

Dans ce mémoire, nous proposons de définir un nouveau langage de modélisation graphique destiné à la construction de modèles de "parts". Les parts sont des objets physiques développés ou produits par les systèmes manufacturiers. Ce langage emprunte ses concepts au pattern *Part* défini dans la thèse de Michaël Petit, *Formal Requirements Engineering of Manufacturing Systems : A Multi-Formalism and Component-Based Approach*, présentée en 1999 à l'Université Notre-Dame de la Paix de Namur, Département de Science informatique. Le pattern *Part* possède un certain nombre de contraintes et est structuré en un ensemble de classes qui entretiennent des relations entre elles.

Nous avons adapté le logiciel Microsoft Visio pour construire les modèles graphiques de "parts". Nous employons une technique de réutilisation qui consiste à instancier ou à spécialiser les classes du méta-modèle du pattern *Part* représentées par des formes. Le logiciel Visio vérifie une partie des contraintes qui s'exercent sur ces modèles, notamment grâce à une application écrite en Visual Basic. Il traduit aussi automatiquement ces modèles et certaines contraintes en types de données et en contraintes AlbertII, langage formel orienté agents employé pour exprimer les spécifications des besoins formels des systèmes manufacturiers. Dans le futur, ce logiciel devra permettre d'enregistrer les modèles graphiques de "parts" afin de pouvoir les réutiliser pour élaborer de nouveaux modèles.

Mots-clés :système manufacturier, part, pattern, classe, réutilisation, orienté agents, AlbertII, éditeur graphique, Visio, Visual Basic for Applications.

In this memory, we propose to define a new language of graphic modeling intended for the construction of models of "parts". The parts are physical objects processed and produced by the manufacturing systems. This language borrows its concepts from the pattern *Part* defined in the thesis of Michaël Petit, *Formal Requirements Engineering of Manufacturing Systems : A Multi-Formalism and Component-Based Approach*, presented in 1999 at the University Notre-Dame de la Paix of Namur, Computer Science Department. The pattern *Part* has a certain number of constraints and is structured in a set of classes that are related to each other.

We adapted the software Microsoft Visio to build the graphic models of "parts". We use a technique of re-use which consists in instantiating or specializing the classes of the meta-model of the pattern *Part* represented by forms. The Visio software checks part of the constraints which are exerted on these models, in particular by means of an application written in Visual Basic. It also automatically translates these models and certain constraints into types of data and constraints AlbertII, a formal agent-oriented language employed to express the specifications of the formal requirements for the manufacturing systems. In the future, this software will have to make it possible to record the graphic models of "parts" in order to be able to re-use them to work out new models.

Keywords : manufacturing system, part, pattern, classe, re-use, agent-oriented, AlbertII, graphic software, Visio, Visual Basic for Applications.

Remerciements

La réalisation de notre étude n'aurait pas été possible sans les appuis et conseils de plusieurs personnes.

À ce titre, nous tenons particulièrement à remercier notre superviseur, Monsieur Michaël Petit, chargé de cours aux Facultés Universitaires Notre-Dame de la Paix à Namur, Département de Science informatique, pour avoir consacré une part de son temps si précieux afin de répondre à nos questions et de guider nos recherches avec pédagogie.

Nous exprimons de même notre gratitude à "Clix" qui a trop généreusement prodigué son temps et sa peine à la correction orthographique de notre travail.

Notre reconnaissance va aussi à notre cousine Françoise et à ses grands talents d'interprète pour avoir eu la gentillesse de nous éclairer sur l'un ou l'autre point de la thèse en anglais qui nous a posé des difficultés de compréhension et de vérifier la traduction du résumé de notre mémoire dans la langue de Shakespeare.

Nous tenons, d'autre part, à remercier Monsieur Vincent Englebert, chargé de cours aux Facultés Universitaires Notre-Dame de la Paix à Namur, Département de Science informatique, pour l'amicale obligeance avec laquelle il nous a prêté la documentation relative au logiciel Microsoft Visio.

Nous prions Madame Anne-Marie Breny, secrétaire aux études de l'Institut d'informatique, de bien bouloir trouver ici l'expression de notre vive reconnaissance pour sa gentillesse et son dévouement.

Enfin, nous tenons à exprimer toute notre gratitude à nos parents, à notre sœur, et à tous ceux que nous n'avons pas cités, pour leur soutien moral et leur patience tout au long de la réalisation de notre travail.

Table des matières

Table des figures

Table des tableaux

Glossaire

AlbertII : langage formel orienté agents pour la modélisation des besoins des systèmes distribués en temps réel. Il est l'un des trois langages employés pour exprimer les besoins des systèmes manufacturiers.

Contraintes AlbertII : formules prédéfinies en AlbertII qui sont des déclarations logiques identifiant les comportements possibles d'un système composite et excluant ceux qui sont indésirables.

Sous-langage AlbertII : sous-ensemble du langage AlbertII utilisé pour exprimer les types de données et les contraintes traduites des modèles de "parts" construits à partir du sous-ensemble du pattern *Part*.

Types de données AlbertII : ensemble des variables d'un jeu de données et des opérations permises sur ces variables. [Dictionnaire 2001] Elles sont contenues dans la partie déclaration d'une spécification AlbertII et introduisent le vocabulaire de l'application considérée.

CIMOSA : langage consacré aux systèmes manufacturiers dont il fournit la représentation des raisons fonctionnelles et des propriétés de comportement. Il est le principal des trois langages employés pour exprimer les besoins des systèmes manufacturiers.

Classe : en programmation orientée objets, la classe est un modèle abstrait définissant des variables et des méthodes pour un type donné d'objet, et à partir duquel sont créés des objets concrets possédant des valeurs particulières. [Dictionnaire 2000] Autrement dit, la classe est la description d'un ensemble d'objets ayant une structure de données commune et disposant des mêmes méthodes, les objets apparaissant comme des variables d'un tel type classe (on dit aussi qu'un objet est une instance de sa classe). [Delannoy 2003, p. 7]

Classe d'attributs : une classe d'attributs regroupe les attributs d'un type particulier d'une classe de parts ou d'états. Elle est elle-même l'attribut de cette classe et elle définit le type d'attributs que les instances de cette classe peuvent avoir.

Classe d'états : classe d'identités, de position ou de dispositifs (relatifs à l'agencement et à l'aspect physique d'une classe de parts) dont les instances caractérisent les états des instances d'une classe de parts.

Classe générale : instance d'une méta-classe qui définit les éléments communs à toutes les instances (classes spécifiques) d'une méta-classe définie au niveau d'un pattern. Ces éléments décrivent les propriétés communes à toutes les occurrences de part de ces classes spécifiques.

Classe de parts : une classe de parts définit l'ensemble des propriétés de toutes les occurrences de part que cette classe possède (comme instances).

Classe spéciale : instance d'une méta-classe et spécialisation d'une classe générale qui définit les éléments communs à toutes les instances (classes spécifiques) d'une

méta-classe définie au niveau d'un pattern. Ces éléments décrivent les propriétés communes à toutes les occurrences de part de ces classes spécifiques. Les classes spéciales sont inclues dans le pattern *Part* et son sous-ensemble parce qu'elles apparaissent souvent dans les systèmes manufacturiers.

Classe (spécifique) : classe obtenue par instanciation d'une méta-classe et par spécialisation d'une classe générale ou d'une classe spéciale d'un pattern dont elle hérite des propriétés (attributs, contraintes et usages). Elle représente une classe nécessaire à l'application. Elle peut être une classe d'attributs, d'états ou de parts.

Méta-classe : une méta-classe d'un pattern spécifique à un domaine sert à définir les classes spécifiques d'une application au moyen du mécanisme d'instanciation.

i* : structure qui vise la modélisation des raisons qui sous-tendent les structures organisationnelles. Il est l'un des trois langages employés pour exprimer les besoins des systèmes manufacturiers.

Instanciation : en programmation orientée objets, création d'un objet sur le modèle de la classe à laquelle il appartient. [Dictionnaire 2002] Un objet lié à une classe par une liaison d'instanciation est appelé une instance de cette classe.

Mappage : transformation de données consignées dans un format en un format différent en associant des données appartenant à un ensemble (champ source) avec les données appartenant à un autre ensemble (champ cible), de manière que les données du premier ensemble puissent se substituer à celles du second ensemble. [Dictionnaire 2001]
Le mémoire fait référence à trois familles de règles de mappage :

- la première établit une correspondance entre les concepts des trois formalismes AlbertII, i* et CIMOSA employés pour exprimer les besoins des systèmes manufacturiers.
- la deuxième traduit un ensemble de patterns en un modèle correspondant en langages AlbertII, i* et CIMOSA.
- la troisième établit une correspondance entre les concepts du sous-ensemble du pattern *Part* et du sous-langage AlbertII définis dans le mémoire, permettant la traduction des modèles graphiques de "parts" en langage AlbertII.

Modèle graphique de "parts" : schéma réalisé à l'aide d'un éditeur graphique Visio adapté et constitué de formes réutilisées et personnalisées représentant des classes spécifiques et les liaisons entre ces classes qui sont elles-mêmes réutilisées par instanciation des méta-classes et par spécialisation des classes générales et spéciales du sous-ensemble du pattern *Part*. Ces classes définissent les parts d'un système manufacturier particulier et sont traduisibles en types de données et en contraintes AlbertII.

Multi-formalisme : approche qui consiste à employer plusieurs formalismes pour construire le modèle d'un système. Un formalisme sert à représenter des connaissances, ses objets et leurs attributs constituant les éléments de base de la connaissance. [Dictionnaire 1994]

Occurrence d'une entité : entité spécifique d'un type d'entités donné. [Dictionnaire 2004] Par exemple, Namur est une occurrence de l'entité ville. Le terme "occurrence" est plutôt utilisé dans des ensembles de données de même nature. Par exemple, les occurrences de part forment l'ensemble de toutes les parts.

Orienté agents : concept basé sur la réutilisation de composants génériques, dont les constituants sont des agents composés de leurs données définitoires et de leurs procédures de manipulation, afin de construire ou de modéliser des systèmes cohérents. Un agent est une entité physique ou virtuelle possédant des ressources propres, capable de percevoir son environnement, d'agir sur lui, de communiquer directement avec d'autres agents et dont les comportements visent à satisfaire ses propres objectifs. [Dictionnaire 2002].

Contrainte : instance de la méta-classe *PatternConstraint* qui est utilisée avec d'autres contraintes pour restreindre l'ensemble des comportements possibles d'un agent ou d'une classe d'agents en un certain nombre de comportements admissibles.

Part : objet physique destiné à être développé et produit par les systèmes manufacturiers depuis les matières premières jusqu'aux produits finis. Un boulon, une planche, un fil de fer, un circuit électronique, un pneu, une voiture ou un téléviseur, par exemple, peut être une part. Une part est aussi l'instance d'une classe de parts.

Attribut : liaison nommée qui permet de définir certains états ou parts comme propriétés des autres. Elle est l'instance d'une classe d'attributs.

Cardinalité : intervalle d'entiers dont les bornes inférieures et supérieures indiquent respectivement le nombre minimum et le nombre maximum d'attributs qu'un état ou qu'une part peut posséder.

Conteneur : dans le sous-ensemble du pattern *Part*, le conteneur est une part composée spéciale dont les contenus sont contigus entre eux et peuvent aussi être enclos. Au sein du pattern *Part*, il est une part composée spéciale qui contient une pile et qui détermine si cette pile est ouverte à l'une ou aux deux extrémités.

État : position et structure (composants, agencement et aspect physique) de la part à un moment donné. Un état est aussi l'instance d'une classe d'états.

Identité : l'identité d'une part permet de distinguer cette part des autres parts existantes. Dans le sous-ensemble du pattern *Part*, elle est dérivée d'un ensemble d'identifiants (un identifiant est une instance de la classe spécifique d'identités). Au sein du pattern *Part*, elle peut aussi être définie par tout ou partie de l'état d'une part.

Part de base : part atomique, qui n'est pas constituée d'autres parts.

Part composée : agrégat formé d'un ensemble d'autres parts.

Pile de parts : part composée spéciale dont les composants sont placés les uns sur les autres.

Tampon : part composée spéciale dans le pattern *Part* et simple part dans son sous-ensemble qui est définie comme un endroit où un certain nombre de parts peuvent demeurer quelque temps.

Pattern : modèle générique qui contient des composants réutilisables destinés à modéliser les aspects physiques des systèmes manufacturiers et qui correspond à la connaissance sur la manière de modéliser de tels systèmes. Il est structuré en un ensemble de classes qui entretiennent des relations entre elles et avec les classes issues d'autres patterns.

Pattern *Part* : graphe constitué d'un ensemble de nœuds appelés méta-classes, classes générales et classes spéciales et d'un ensemble de liaisons définies en attributs, spécialisations et instanciations. Le pattern *Part* est défini dans la thèse de Michael Petit. [Petit 1999, pp. 183-204]

Sous-ensemble du pattern *Part* : pattern redéfinissant les méta-classes, les classes générales et les classes spéciales du pattern *Part* pour les adapter à la modélisation graphique dans Visio. Il est défini dans ce mémoire et est utilisé pour construire les modèles de "parts".

Spécialisation : liaison particulière entre les classes qui signifie une inclusion de toutes les instances d'une classe (la sous-classe) dans une autre classe (la super classe).

Héritage : concept selon lequel l'instance d'une sous-classe possède les attributs de la super classe dont elle est aussi automatiquement une instance.

Partition : une super classe abstraite avec un ensemble de sous-classes exclusives.

Sous-classes exclusives : une instance de la super classe ne peut être dans plus d'une des sous-classes en même temps.

Super classe abstraite : toutes les instances de la super classe doivent être dans au moins une des sous-classes.

Syntaxe : pour un langage de programmation, la syntaxe est l'ensemble des règles déterminant l'écriture et la disposition des instructions d'un programme, de telle sorte que ce langage soit accessible au compilateur ou à l'interpréteur. [Dictionnaire 2000]

Syntaxe abstraite : spécification de données de la couche application au moyen de notations indépendantes de la technique de codage utilisée pour représenter ces données. [Dictionnaire 1992]

Syntaxe concrète : aspects des conventions utilisées dans la description formelle des données qui recouvrent une représentation spécifique de ces données. [Dictionnaire 1992]

Système manufacturier : série de processus définissant une manufacture et destinés à transformer une matière première en formes plus utilisables et éventuellement en produits finis commercialisables.

Telos : langage formel utilisé pour définir la combinaison des langages AlbertII, i* et CIMOSA employés pour exprimer les besoins des systèmes manufacturiers.

Visio : éditeur graphique de la suite Microsoft Office. Un éditeur graphique est un programme interactif permettant de faire certaines opérations sur un dessin telles que des rotations, des translations ou des symétries. [Dictionnaire 1987]

Visual Basic for Applications : langage de programmation qui facilite la rédaction de programmes par l'utilisation de combinaisons de commandes et d'instructions préétablies. Il permet de personnaliser et d'ajouter des fonctionnalités à l'application pour laquelle il est défini (Visio dans le mémoire). [Dictionnaire 2002]

Introduction

Le langage de spécification orienté agents AlbertII a été appliqué à la modélisation des systèmes de production manufacturiers. Ces systèmes combinent des composants logiciels, des agents humains et des composants industriels (machines). Étant donné les similitudes existant entre la plupart de ces systèmes, des modèles des exigences de tels systèmes peuvent généralement être élaborés à partir de composants de spécifications génériques (réutilisables), rapidement instanciables.

Le mémoire a pour objet de spécifier et de construire l'embryon d'un outil logiciel d'édition de spécifications AlbertII dédié aux systèmes de production afin de déterminer dans quelle mesure cet outil pourrait permettre l'instanciation de composants génériques. L'objectif poursuivi à terme est d'enregistrer les composants réutilisables et la spécification obtenue après réutilisation pour qu'ils servent de base à la future plate-forme d'outils.

Le mémoire se situe dans le prolongement de la thèse de Michaël Petit, *Formal Requirements Engineering of Manufacturing Systems : A Multi-Formalism and Component-Based Approach*, présentée en octobre 1999 à l'Université Notre-Dame de la Paix à Namur, Département de Science informatique, en vue de l'obtention du grade de Docteur en Philosophie. La thèse de Michaël Petit propose un cadre de travail pour modéliser les exigences des systèmes manufacturiers dans une approche qui se veut multi-formaliste et orientée vers la réutilisation de composants. La technique de réutilisation est basée sur une série de patterns qui modélisent les aspects physiques des systèmes manufacturiers. Chaque pattern est structuré en un ensemble de classes qui sont associées entre elles. L'un d'entre eux, le pattern *Part*, est utilisé pour construire les modèles de "parts". Les parts sont des objets physiques destinés à être développés et produits par les systèmes manufacturiers. Les concepts de ce pattern sont exprimés en langage AlbertII et plus particulièrement en types de données AlbertII.

Dans ce mémoire, nous allons définir un nouveau langage de modélisation basé sur le pattern *Part* et adapté à un éditeur graphique appelé aussi outil logiciel. Celui-ci devra exprimer visuellement les concepts du pattern *Part* repris dans ce langage et adaptés à ce mode d'expression particulier. Nous escomptons aussi faciliter la conception des spécifications des systèmes manufacturiers en adoptant une modélisation graphique.

L'outil logiciel présentera quatre principales fonctionnalités. La première permettra à l'utilisateur de dessiner des modèles de "parts". La deuxième vérifiera une partie des contraintes qui s'exercent sur ces modèles et qui se rapportent soit au langage de modélisation, soit aux caractéristiques de la modélisation graphique. La troisième traduira automatiquement ces modèles et certaines contraintes en types de données et en contraintes AlbertII dans une syntaxe correcte. La quatrième et dernière fonctionnalité ne sera pas implémentée. Elle consistera à enregistrer les modèles graphiques de "parts" dans le but, d'une part, de réutiliser des modèles antérieurs d'un système existant pour modifier celui-ci (de manière incrémentale) et, d'autre part, d'élaborer de nouveaux modèles de systèmes au moyen de modèles génériques, ce que ne permettent pas les modèles construits en langage AlbertII.

La première et principale source d'informations provient de la thèse de Michael Petit. Celle-ci décrit notamment le langage AlbertII employé pour exprimer les spécifications des besoins formels des systèmes manufacturiers. Elle définit également une modélisation de ces systèmes basée sur les patterns et plus particulièrement sur le pattern *Part*. La seconde source d'informations est le logiciel Visio développé par la firme Microsoft et qui sert de plate-forme pour l'édition graphique de modèles de "parts" et pour leur traduction en langage AlbertII. Outre une importante documentation, il comprend un kit de développement utilisant le langage Visual Basic qui permet de créer des applications spécifiques.

Le mémoire est partagé en trois chapitres. Le premier expose brièvement les concepts développés dans la thèse et qui font l'objet de notre étude. Il débute par une présentation des systèmes manufacturiers et de l'approche qui sous-tend la thèse. Il aborde ensuite le langage AlbertII et se termine par une définition du pattern *Part*.

Le deuxième chapitre est consacré au logiciel Visio. Il commence par une présentation générale de l'éditeur graphique. Puis il expose les raisons pour lesquelles ce logiciel a été choisi afin de modéliser graphiquement les définitions de "parts". La dernière partie du chapitre passe en revue les différents concepts de Visio qui sont utilisés dans le cadre de cette étude.

Le troisième chapitre qui est le plus volumineux représente le cœur du mémoire : la création de modèles graphiques de "parts", la vérification de leurs contraintes et leur traduction en langage AlbertII. La première partie est consacrée au langage de modélisation graphique avec une redéfinition des classes du pattern *Part*, l'établissement de correspondances entre les concepts de ce langage et ceux du logiciel Visio, l'élaboration de règles de réutilisation de ces classes pour construire des modèles, et la présentation des contraintes qui sont vérifiées par l'outil logiciel, qui sont traduites en contraintes AlbertII ou qui sont invérifiables et intraduisibles. La deuxième partie du chapitre développe un sous-langage AlbertII dont elle expose les caractéristiques principales et la syntaxe concrète (en annexe). La troisième et dernière partie présente les règles de mappage qui président à la traduction des modèles graphiques de "parts" en types de données et contraintes AlbertII.

Les annexes comprennent le méta-modèle complet du langage de modélisation basé sur le pattern *Part*, les formes employées par l'éditeur Visio pour représenter les classes de ce langage, la syntaxe concrète du même langage, un exemple de modélisation graphique de "parts" emprunté à la description de trois modèles de tracteurs Massey Ferguson, un exemple d'application des règles de traduction des modèles graphiques en langage AlbertII, et le code VBA (Visual Basic for Applications) associé à l'outil logiciel.

Au cours de l'exposé, nous formulerons des remarques critiques et constructives à l'égard de certains concepts définis dans la thèse, notamment relatifs au pattern *Part*. Nous tenterons également d'y apporter quelques corrections ou améliorations tout en conservant la philosophie générale de la thèse. Nous privilégierons l'aspect pratique dans la conception du langage de modélisation et de l'outil logiciel car l'objectif est de réaliser un outil utilisable. Nous essayerons aussi d'être didactique en n'étant pas avare d'explications et en proposant de nombreux exemples dont un complet de modélisation graphique en annexe D.

La police de caractères *Times New Roman italique* sera employée pour écrire les noms des classes, tandis que les éléments de code AlbertII seront rédigés en `Courier New`. La traduction française la plus proche de "part(s)" est le mot "pièce(s)". Mais pour marquer la filiation avec la thèse, nous préférons conserver le terme anglais dont la prononciation française en alphabet phonétique international est, au singulier ou au pluriel, **[paRt]**.

Chapitre 1 :

Présentation de l'approche pour une modélisation des besoins formels des systèmes manufacturiers

Dans ce chapitre, nous exposerons brièvement les concepts développés dans la thèse de Michaël Petit [Petit 1999] qui sont liés au mémoire. Nous commencerons d'abord par une présentation de l'approche qui sous-tend la thèse et qui se caractérise par un multi-formalisme et une technique basée sur la réutilisation de composants. Elle sera suivie par la définition des systèmes manufacturiers auxquels elle se rapporte. Nous aborderons ensuite le langage AlbertII qui constitue l'un des trois langages employés pour exprimer les besoins des systèmes manufacturiers. Nous terminerons enfin par le pattern *Part* qui sert de cadre de référence pour la modélisation des systèmes manufacturiers et plus particulièrement des objets physiques destinés à être développés et produits par ces systèmes.

1.1. Présentation générale de l'approche et des systèmes manufacturiers

1.1.1. L'approche

Dans sa thèse, Michaël Petit propose un cadre de travail[1] pour la modélisation des besoins des systèmes manufacturiers à l'attention des ingénieurs de ce domaine destiné à soutenir leur travail de modélisation dans un environnement en constante évolution. Il se base sur deux principes importants : une approche multi-formaliste, combinant plusieurs langages dans un formalisme cohérent, et une approche de modélisation fondée sur les composants, arguant que les modèles des systèmes manufacturiers peuvent être partiellement construits par l'assemblage et l'adaptation de composants génériques issus d'un modèle réutilisable.

L'approche multi-formaliste consiste à employer plusieurs formalismes pour construire le modèle d'un système. La structure de modélisation proposée combine les langages AlbertII, i* et CIMOSA qui sont complémentaires en raison de leurs différents degrés d'expressivité, de naturel et de précision. Dans le cadre du mémoire, notre intérêt se portera uniquement sur le langage AlbertII qui est un langage formel orienté agents pour la

[1] Un cadre de travail fournit habituellement des solutions presque complètes pour un domaine d'application particulier. Il est défini dans la thèse aussi bien comme la conception réutilisable de tout ou partie d'un système qui est représenté par un ensemble de classes abstraites et par la manière dont les instances de ces classes interagissent, que comme le squelette d'une application qui peut être adaptée par un développeur d'application. [Petit 1999, p. 162]

modélisation des besoins des systèmes distribués en temps réel.[2] CIMOSA est, quant à lui, un langage consacré aux systèmes manufacturiers tandis que i* est une structure qui vise la modélisation des raisons qui sous-tendent les structures organisationnelles.[3]

Le cadre de travail s'appuie également sur la réutilisation de composants en fournissant une série de patterns pour la modélisation des aspects physiques des systèmes manufacturiers. Chacun d'entre eux est structuré comme un ensemble de classes qui entretiennent des rapports les unes avec les autres et avec les classes issues d'autres patterns. Les patterns peuvent être réutilisés et adaptés pour l'application. Ils sont d'abord construits indépendamment des trois langages puis un ensemble de règes de mappage sont définies pour obtenir un modèle correspondant dans les trois langages.

1.1.2. Les systèmes manufacturiers

La manufacture est définie dans la thèse comme une activité organisée dévolue à la transformation de matières premières en produits commercialisables qui peut être vue comme une série de processus manufacturiers à valeur ajoutée destinés à transformer une matière première en formes plus utilisables et éventuellement en produits finis. [Petit 1999, p. 19] Ces processus, qui peuvent être considérés comme un système manufacturier, reçoivent notamment en entrée des objets de production, de la main d'œuvre, des moyens de production et de l'information. Les biens produits en sortie peuvent être utilisés par des consommateurs qui sont parfois eux-mêmes des manufacturiers.

Les fonctions réalisées dans un système manufacturier incluent la détermination des objectifs commerciaux, le marketing et la vente, la planification (la détermination des objectifs à long terme de l'entreprise), la conception du produit, l'organisation de la production (définie comme la conception des séquences des opérations nécessaires à la fabrication des produits et à la sélection des ressources pour exécuter ces opérations), la gestion des ressources temporelles et matérielles, la production et l'assemblage, le stockage, l'expédition des produits et la qualité de la gestion. Les systèmes manufacturiers emploient une large variété de ressources qui peuvent être classées en trois catégories : les équipements de production (machines), les technologies de l'information et les ressources humaines.

L'ingénierie des systèmes manufacturiers est définie comme l'activité de construction ou de modification d'une partie ou de l'ensemble d'un système manufacturier d'une manière organisée et systématique dans le but d'assurer l'adéquation du futur système avec un ensemble donné d'objectifs. [Petit 1999, p. 20] Le processus d'ingénierie des systèmes manufacturiers implique un certain nombre d'activités séquentielles – bien que certaines puissent être menées en parallèle – qui sont souvent représentées sous la forme d'un système de cycle de vie : l'analyse de la situation courante, la définition des besoins relatifs au problème ou à la nouvelle opportunité, la conception d'un système capable d'y répondre, l'implantation, la mise en fonction et la maintenance de ce système, jusqu'à son démantèlement. Toutefois, dans notre mémoire, nous nous intéresserons uniquement à quelques concepts proposés pour les phases d'analyse des besoins et de conception préliminaire.

---

[2] AlbertII vise la modélisation des propriétés fonctionnelles et comportementales des systèmes. Il est beaucoup plus précis que CIMOSA et i* qui proposent parfois des concepts avec une pauvreté sémantique.

[3] CIMOSA fournit la représentation des raisons fonctionnelles et des propriétés de comportement mais possède des concepts qui conviennent particulièrement à la manufacture, tandis que i* permet l'expression des propriétés rationnelles et non fonctionnelles.

1.2. Le langage AlbertII

Le langage AlbertII est un langage formel de spécification des besoins logiciels, c'est-à-dire qu'il vise la modélisation du rôle du système du futur logiciel en terme d'interactions avec l'environnement. Il fournit ainsi une vérification et une validation puissante du document des besoins. Sa conception débute à l'université de Namur vers 1992. Il est d'abord employé dans le contexte des systèmes non triviaux comme le Computer Integrated Manufacturing ou les systèmes de télécommunication. Il est donc plus spécifiquement un langage orienté agents pour la construction et la découverte des besoins des systèmes en temps réel.

1.2.1. Les caractéristiques du langage AlbertII

L'activité d'ingénierie des besoins requiert la spécification de langages expressifs et naturels. Un langage expressif est un langage qui offre une ontologie[4] de concepts suffisamment riche pour modéliser le domaine d'application étudié. En pratique, l'expressivité du langage AlbertII dépend de concepts capables de capturer les aspects temps réel, la distribution et la communication entre les agents autonomes. Un langage utilisé dans le cadre de l'ingénierie des besoins doit aussi être proche du langage courant : les besoins exprimés par les clients doivent être facilement traduits sous forme de constructions de langage sans devoir les encoder dans un style opérationnel exécutable (au risque d'introduire des spécifications supplémentaires). Le caractère naturel du langage AlbertII provient du style déclaratif qu'il supporte pour l'expression des contraintes.

Le langage AlbertII est entièrement formel. Un langage formel est un langage équipé d'une sémantique mathématique et logique adéquate constituée, d'une part, de règles d'interprétation qui garantissent l'absence d'ambiguïtés dans le document des besoins et, d'autre part, de règles de déduction qui permettent de raisonner sur le document de spécification dans le but de découvrir les carences, les incohérences potentielles ou de démontrer les propriétés.

Le langage AlbertII est fondamentalement basé sur une variante de la logique temporelle en temps réel appelée Albert-KERNEL, un langage mathématique particulièrement adapté pour décrire les histoires (les séquences d'états) et exprimer les contraintes de fonctionnement (comme, par exemple, "cette propriété dure au moins trois minutes"). Dans le cadre de l'analyse des besoins des systèmes manufacturiers, trois grandes extensions expressives ont été définies :

- les actions : les actions sont associées aux changements qui peuvent modifier les états dans les histoires. L'utilisation des actions en AlbertII permet de résoudre le problème bien connu du frame;
- les agents : un agent est caractérisé par, premièrement, son état interne qui indique la connaissance de l'agent sur son environnement (y compris lui-même), deuxièmement, sa perception de ce qui se passe dans son environnement, et, troisièmement, sa responsabilité dans le respect des actions qui provoquent des effets sur son état ou sur l'état des autres agents;

[4] Ontologie : ensemble d'informations dans lequel sont définis les concepts utilisés dans un langage donné et qui décrit les relations logiques qu'ils entretiennent entre eux. [Dictionnaire 2002]

- les patterns de contraintes : l'utilisation d'un langage formel peut être comparée à l'utilisation d'un langage de programmation assembleur en raison du manque de support pour l'analyste dans la rédaction de déclarations complexes et cohérentes. Pour résoudre ce problème un certain nombre de modèles de formules ont été prédéfinis en AlbertII.

L'utilisation du langage AlbertII induit deux activités : d'abord, l'écriture de déclarations introduisant le vocabulaire de l'application considérée et, ensuite, l'expression de contraintes, c'est-à-dire de déclarations logiques qui identifient les comportements possibles du système composite et excluent ceux qui sont indésirables. Pour la partie déclaration, AlbertII fournit une syntaxe graphique et textuelle tandis que pour les contraintes, seule une syntaxe textuelle existe. Dans notre mémoire, notre attention portera uniquement sur la déclaration des types de données (qui représentent les biens manipulés et fabriqués dans les systèmes manufacturiers, depuis les matières premières jusqu'aux produits finis) et sur la définition des contraintes y afférentes.

1.2.2. L'écriture des déclarations en AlbertII

La partie déclaration d'une spécification contient les types de données, le groupement des agents dans les sociétés, ainsi que la structure d'états et les actions des agents.

Le langage AlbertII est un langage très typé qui dépend de l'utilisation de types de données abstraites. Il distingue :

- les types de données élémentaires (*BasicType*) prédéfinis : *STRING*, *CHAR*, *BOOLEAN*, *INTEGER*, *RATIONAL* et *DURATION*;
- les types correspondant aux identifiants d'un agent;
- les types élémentaires (*BasicType*) définis par l'utilisateur (pour lesquels aucune structure n'est donnée);
- les types construits (*ConstructedType*) définis par l'utilisateur et qui sont élaborés à l'aide des constructeurs de types (*Cartesian product*, *set*, *sequence*, *union of types*, *bag*, *table* et *enumeration of values*).

Comme le nom de la première catégorie le suggère, ces types sont préétablis dans la spécification et ne doivent pas être déclarés explicitement. Les éléments de la deuxième catégorie sont implicitement déclarés en définissant les agents. Seules les deux dernières catégories doivent être déclarées explicitement.

Un type peut recevoir la valeur supplémentaire *UNDEF* par l'adjonction d'une étoile (*) placée à la fin de son nom. Dans ce cas, les valeurs permises pour le type augmenté sont les valeurs du type original plus la valeur *UNDEF*.

Un type de données (baptisé énumération) peut être défini sur la base d'un autre en déclarant une contrainte (un invariant) qui sélectionne un sous-ensemble des instances du type de base comme les instances du nouveau type.

Des opérations peuvent être définies sur les types de données, principalement pour rendre la spécification plus lisible, mais aussi pour représenter le domaine de connaissance du problème concernant l'information présente dans le système. Elles se présentent sous la forme de fonctions mathématiques qui retournent des valeurs qui doivent respecter certaines contraintes logiques du premier ordre liées à leurs arguments et à leurs résultats. Dans notre mémoire, nous n'aurons pas besoin de définir des opérations de ce type mais nous utiliserons par contre certaines opérations prédéfinies pour les types de données et les constructeurs de types prédéfinis.

```
%BASIC TYPES
Piston
%CONSTRUCTED TYPES
Bloc_moteur = CP [ identité:INTEGER,
                   composants_pistons:Piston,
                   composants_bloc:Bloc]
     WITH \ForAll p/Bloc_moteur :
          (Card(composants_pistons) + Card(composants_bloc)) >= 2
Bloc = CP[ Type:ENUM [Bloc_voiture, Bloc_camion, Bloc_tracteur],
           test:BOOLEAN*]
```

Figure 1.1 : Exemple de types de données et de contraintes définis dans notre cas d'étude

La figure 1.1 montre un exemple de types de données et d'une contrainte relatifs à notre cas d'étude. Elle représente la définition partielle en langage AlbertII d'un bloc moteur (`Bloc_moteur`) qui a une identité et qui est composé de plusieurs pistons (`Piston`) et d'un bloc (`Bloc`). Ce dernier peut être soit un bloc de voiture (`Bloc_voiture`), soit un bloc de camion (`Bloc_camion`), soit un bloc de tracteur (`Bloc_tracteur`). En outre, il peut faire l'objet d'un test.

Le piston correspond au type élémentaire défini par l'utilisateur mais qui ne possède pas de structure. Le bloc moteur et le bloc sont tous deux des types définis par l'utilisateur et construits au moyen d'un produit cartésien composé respectivement de trois et deux champs. Le bloc moteur possède une identité dont la valeur est un type prédéfini : `INTEGER`. Une contrainte lui est associée selon laquelle il doit être constitué d'au moins deux composants. Elle est élaborée à l'aide d'une opération prédéfinie pour les types de données (`Card`) qui renvoie le nombre d'éléments d'un ensemble (ici `Piston` et `Bloc`). Le premier champ du bloc est une énumération : pour chaque instance du type de données bloc, ce champ contient une et une seule des valeurs énumérées. Le second champ indique que le test sur le bloc est facultatif puisque son nom se termine par une étoile, signalant ainsi que sa valeur peut être indéfinie.

Les deux autres éléments de la déclaration de spécification ne nous intéressent pas. Le premier concerne les sociétés d'agents. Les agents sont groupés en sociétés qui peuvent être elles-mêmes groupées pour former de plus grosses sociétés. En pratique, la spécification AlbertII consiste en une hiérarchie dont les feuilles sont les agents et dont les nœuds intermédiaires sont les sociétés. Le second élément de la déclaration a trait à la structure d'états et aux actions des agents. La partie déclarative d'un agent consiste en effet dans la description de la structure de ses états (ensembles de valeurs), dans la liste des actions qu'il peut accomplir et dans l'exportation des liens (liaisons visibles) vers les autres agents.

1.2.3. Les contraintes d'un agent

Les contraintes sont utilisées pour restreindre l'ensemble des comportements possibles d'un agent (ou d'une classe d'agents) en un certain nombre de comportements admissibles. La vie d'un agent est définie comme l'un de ses comportements possibles. Les contraintes donnent des affirmations qui peuvent être vraies pour toutes les vies, restreignant ainsi l'ensemble des vies admissibles. Elles sont exprimées en terme de différents modèles de propriétés disponibles dans le langage. Ces modèles ont été regroupés en quatre familles : les contraintes de base, les contraintes déclaratives, les contraintes opérationnelles et les

contraintes de coopération. Dans notre mémoire, nous n'emploierons que les contraintes déclaratives. Celles-ci groupent tous les modèles de contraintes qui ne sont pas opérationnels. Elles décrivent, souvent de manière non déterministe, les états ou actions qui sont distants dans le temps, à l'opposé des contraintes opérationnelles qui décrivent les actions dans les états adjacents.

Contrairement aux langages de spécification conçus pour les logiciels ordinaires, la sémantique d'AlbertII n'est pas opérationnelle : une vie peut être largement considérée avant qu'elle ne puisse être désignée comme admissible ou non, c'est-à-dire qu'ajouter de nouveaux états et qu'effectuer des modifications (occurrences d'actions) à la fin d'une vie admissible ne donne pas nécessairement une vie admissible. Ceci est dû à la possibilité d'exprimer des contraintes déclaratives.

1.2.4. La sémantique formelle

Comme cela a déjà été mentionné ci-dessus, le langage AlbertII est basé sur une logique temporelle en temps réel appelée Albert-KERNEL. Cette logique permet de donner une sémantique formelle au langage. Cet apport engendre deux conséquences. La première, c'est l'absence d'ambiguïté du langage construit : chaque construction possède en effet une définition unique qui est donnée sous la forme de concepts Albert-KERNEL. La seconde conséquence, c'est la possibilité d'effectuer une analyse approfondie sur la spécification : une sémantique précise permet en effet d'obtenir une analyse plus complexe que la cohérence syntaxique classique et que la vérification de la complétude permise par les langages semi-formels. Par exemple, une vérification de modèle peut être utilisée pour valider la consistance de la spécification, c'est-à-dire l'absence de fragments de spécification conflictuels.

1.2.5. AlbertII et les autres langages

Dans la présentation de l'approche, nous avons mentionné que les modèles des systèmes manufacturiers étaient exprimés dans trois langages différents : AlbertII, i* et CIMOSA. Pour éviter la redondance des formalismes, un ensemble de règles de mappage ont été définies afin d'établir une correspondance entre les concepts des trois formalismes. Par ailleurs, Michaël Petit est parti du principe que le langage central du cadre de travail était CIMOSA et il a émis l'hypothèse que le spécificateur utiliserait presque toujours ce langage. Le rôle des langages AlbertII et i* a donc été réduit à celui de support destiné à rendre le modèle CIMOSA plus précis et plus complet.

Un quatrième langage, appelé Telos, est utilisé dans la thèse pour définir la combinaison des trois principaux langages. Telos est un langage formel initialement conçu pour la modélisation conceptuelle et la méta-modélisation requise dans les environnements CASE (Computer Aided Software Engineering). Il est basé sur un ensemble très limité de concepts mais fournit néanmoins des mécanismes d'abstraction très puissants et fondamentaux nécessaires à la méta-modélisation (instanciation, spécialisation et attribution) avec une sémantique très précise. Dans la thèse, il n'est pas employé directement pour la modélisation des systèmes manufacturiers mais il est plutôt utilisé pour exprimer les méta-modèles de tous les langages et pour formaliser la notion de règle de mappage entre ces langages.

La présentation du pattern *Part* à la section suivante nécessite auparavant la définition de certains concepts liés à la description graphique des modèles Telos. Le pattern *Part* se présente en effet sous la forme d'un graphe constitué d'un ensemble de nœuds appelés individus et d'un ensemble de liaisons (ou liens) classées en attributs, spécialisations et instanciations. Voir en exemple la figure 1.2.

Les attributs sont des liaisons nommées qui permettent de définir certains individus comme propriétés des autres. Ils sont représentés par des arcs munis d'une étiquette. Celle-ci comprend le nom de l'attribut, ainsi que la cardinalité de la liaison, c'est-à-dire un intervalle d'entiers dont les bornes inférieures et supérieures indiquent respectivement le nombre minimum et le nombre maximum d'attributs qu'un individu peut posséder.

Les instanciations signifient que les individus qui font partie des autres individus sont considérés comme des classes. Un individu lié à une classe par une liaison d'instanciation est appelé une instance de cette classe. Une classe possède un certain nombre d'attributs qui définissent les types d'attributs que ses instances peuvent avoir. Les attributs de la classe sont des classes d'attributs et les attributs des instances sont les instances de ces classes. Un individu peut appartenir à plus d'une classe et, dans ce cas, il peut posséder des attributs qui sont des instances des classes d'attributs attachées à chacune de ses classes. Les classes peuvent être elles-mêmes des instances des autres classes qui sont appelées méta-classes.

Les spécialisations sont des liaisons particulières entre les classes qui signifient une inclusion de toutes les instances d'une classe (la sous-classe) dans une autre classe (la super classe). Elles sont représentées par des flèches épaisses sur la figure. L'appartenance d'un individu à la sous-classe implique que cet individu est automatiquement une instance de la super classe et que, par conséquent, il peut posséder comme attributs tous ceux permis aux instances de la super classe. Ce mécanisme peut être considéré comme un héritage des attributs de la super classe par la sous-classe. Telos et AlbertII supportent aussi l'héritage multiple : une classe peut être la sous-classe de plusieurs classes.

Les liaisons ne sont pas restreintes aux individus : elles peuvent également se rapporter à d'autres liaisons, permettant ainsi la définition d'attributs attachés à des attributs, la définition de classes et d'instances d'attributs (comme expliqué ci-dessus) et la définition de liaisons de spécialisation entre attributs.

1.3. Le pattern *Part*

1.3.1. Une modélisation basée sur les patterns

Outre son caractère multi-formaliste, le cadre de travail présente un ensemble de modèles génériques, baptisés patterns, qui peuvent être utilisés pour définir certains aspects des systèmes manufacturiers. Ces patterns correspondent à la connaissance sur la manière de modéliser un tel système. Chacun d'entre eux est structuré en un ensemble limité de classes qui possèdent des relations les unes avec les autres.[5] Ils peuvent être facilement adaptés pour l'application en cours. À l'instar des langages orientés objets, la réutilisation d'un pattern consiste à sélectionner un sous-ensemble de ces classes et à les spécialiser pour représenter les classes nécessaires à l'application. Les classes prédéfinies étant associées à un nombre de propriétés (incluant des contraintes) qui sont héritées grâce au processus de spécialisation, ces propriétés ne doivent pas être redéfinies pour chaque nouvelle application. De plus, l'analyste

[5] Un pattern contient généralement un maximum de cinq classes.

a la possibilité de définir ses propres patterns (en ajoutant des propriétés spécifiques) pour une réutilisation ultérieure.

Les patterns sont d'abord exprimés indépendamment des langages du cadre de travail et un ensemble de règles de mappage similaires à celles entre les trois formalismes sont définies pour obtenir un modèle correspondant en AlbertII, i* et CIMOSA. Cette méthode a l'avantage de pouvoir traduire un modèle généré par l'instanciation du méta-modèle de pattern dans n'importe quel langage. Pour cela, des règles de mappage spécifiques à chaque pattern ont été définies sur base des concepts présents au niveau du méta-modèle.

Les patterns contribuent à accélérer le processus de modélisation de deux manières : d'abord, par la réutilisation qui évite à l'analyste de devoir redéfinir des spécifications déjà présentes dans un pattern existant, ensuite, par l'utilisation de concepts de haut niveau spécifiques au domaine qui sont, par voie de conséquence, fort compréhensibles par les ingénieurs de l'entreprise.

1.3.2. Le pattern *Part*

Le pattern *Part* est utilisé pour construire les modèles de "parts". Comme la plupart des patterns, il possède une structure composée de deux types d'éléments distincts : un ensemble de méta-classes spécifiques au domaine pour définir les classes spécifiques à l'application (des classes de parts obtenues par instanciation) et un ensemble de classes générales spécifiques au domaine d'où proviennent les éléments communs (qui décrivent les propriétés communes à toutes les occurrences de part) à toutes les classes spécifiques de l'application.

1.3.2.1 Les classes générales du pattern *Part*

Les classes générales d'un pattern représentent les caractéristiques universelles qui s'appliquent à toutes les instances d'une classe d'objets dans une application. Par exemple, la classe générale *CompoundPart* du pattern *Part* groupe les dispositifs qui s'appliquent universellement aux parts composées (parts faites d'autres parts comme composants).

Les occurrences de part (ou simplement "parts") sont les objectifs finaux des systèmes manufacturiers. Ils sont considérés ici comme des objets physiques destinés à être développés et produits par ces systèmes. Ceux-ci incluent les produits finaux, les composants, les matières premières et même les outils s'ils sont développés d'une certaine manière par le système manufacturier (par exemple s'ils sont assemblés, transportés ou conservés dans des stocks).

Le pattern *Part* définit : d'une part, les caractéristiques obligatoires (attributs et contraintes) qui s'appliquent à toutes les parts d'une application (indépendamment de la classe spécifique à laquelle elles peuvent appartenir) et qui doivent nécessairement être présentes dans chaque modèle, et, d'autre part, les simples propriétés optionnelles qui sont fréquemment utilisées dans les modèles de "parts".

Certaines parts possèdent le même ensemble de caractéristiques mais chaque part dispose d'une identité (*identity*) différente de toutes les autres parts. Les caractéristiques d'une part peuvent être modifiées mais son identité reste toujours la même. La notion d'états (*states*) d'une part définit ce principe : durant sa vie, une part avec une identité donnée peut changer d'état plusieurs fois mais à tout moment elle est associée à un et un seul état.

L'état d'une part est constitué :

- d'un état position (information sur la position de la part qui respecte une position fixée par l'atelier de fabrication avec, par exemple, les coordonnées de la part dans l'espace);
- d'un état structurel composé lui-même :
  - d'un état matériel décrivant l'ensemble des composants constituant la part (qui sont eux-mêmes des parts avec un état donné);
  - d'un ensemble de dispositifs géométriques définissant les liens géométriques existant entre les composants et représentant la manière dont ils se combinent pour former une seule part (comme, par exemple, les contacts au niveau des surfaces);
  - d'un ensemble de spécifications représentant la manière dont les dispositifs géométriques sont fixés (au moyen de colle, de rivets, de vis, etc.) et qui sont appelées dispositifs de fixation;
  - d'un ensemble de spécifications représentant les aspects physiques des parts tels que les caractéristiques manufacturières (calibre des trous, surfaces crénelées, etc.) ou les dispositifs complémentaires (comme les surfaces peintes, les résultats de tests, etc.) et qui sont appelées dispositifs physiques.

Une distinction est faite entre les parts de base et les parts composées. Une part composée est un agrégat formé d'un ensemble d'autres parts qui peuvent être elles-mêmes décomposées en d'autres parts. Les parts de base sont les parts atomiques, soit celles qui ne sont pas constituées d'autres parts. Les parts de base diffèrent des parts composées parce qu'elles ne possèdent ni état matériel, ni dispositifs géométriques, ni dispositifs de fixation. Elles peuvent seulement posséder une identité, un état position et des dispositifs physiques. L'état position et les dispositifs géométriques des composants ne sont pas développés dans la thèse : ils sont supposés être définis par un ensemble de valeurs nommées qui représentent leurs positions (ou leurs contacts) possibles.

Une distinction est également faite entre les dispositifs de part simples et les dispositifs de part évalués :

- un dispositif de part simple (*simple part feature*) est un dispositif qu'une ou plusieurs parts peuvent avoir à un moment donné. Par exemple, "noir" est le nom d'un dispositif physique qu'une part peut avoir pour indiquer sa couleur;
- un dispositif de part évalué (*valued part feature*) est un dispositif qu'une part possède ou non. Mais si la part possède le dispositif, celui-ci doit être associé à une ou plusieurs valeurs associées qui donnent une information sur lui. Par exemple, "calibre_des_trous" est le nom d'un dispositif physique d'une part qui renseigne qu'un trou a été foré dans la part et qui est toujours défini par deux valeurs indiquant respectivement la profondeur et le diamètre du trou.

Au niveau des classes générales, cette différence se traduit par le fait qu'une classe de dispositifs de valeur n'est jamais associée à une classe de dispositifs simples tandis qu'une au moins est toujours attachée à une classe de dispositifs évalués par l'intermédiaire d'une classe d'attributs valeurs dont la borne inférieure de la cardinalité ne peut être nulle.

Comme mentionné ci-dessus, l'identité représente une caractéristique importante de la part : si son état peut évoluer avec le temps, son identité reste toujours la même. Les parts peuvent être créées ou détruites (perte de leur identité). L'identité d'une part permet de distinguer celle-ci des autres parts existantes. Elle peut éventuellement être dérivée de l'état de la part si deux parts ne peuvent pas être dans le même état au même moment. Une partie de l'état d'une part peut également être utilisée pour la différentier des autres parts. Cette

partie inclut aussi bien sa position, les dispositifs qu'elle possède, son identité, qu'un ou plusieurs de ses composants.

Si l'état d'une part permet de la distinguer des autres parts, aucune information supplémentaire que celle présente pour décrire son état n'est nécessaire. Dans le cas contraire, un identifiant doit être attaché à la part. Il représente une information supplémentaire (un identifiant de part qui est une instance de la classe d'identités) qui peut appartenir à différentes classes (entier, string, etc.). Cette information additionnelle est alors utilisée pour différencier la part.

Il existe une relation entre la position d'une part composée, ses dispositifs géométriques et la position de ses composants. Toutefois, ce lien n'est pas défini dans la thèse.

À un moment donné dans le temps, un ensemble d'occurrences de part forme un ensemble de parts existantes dans le système considéré. L'existence ou non de parts n'est qu'une vue de l'esprit : des parts existant dans le monde réel peuvent être considérées comme n'existant pas dans le modèle, tandis que des parts qui ne sont pas encore présentes dans le monde réel peuvent voir leur existence déjà prise en compte (c'est-à-dire être définie comme une part produite). L'ensemble des parts existant à un moment donné peut être défini comme un sous-ensemble de l'ensemble des parts (concevables). Ce principe est consacré en définissant la classe des parts existantes comme un sous-ensemble de la classe de toutes les parts. Sur l'ensemble des parts existantes, deux parts ne peuvent posséder la même identité. En outre, les contraintes physiques imposent qu'une part ne puisse être simultanément le composant de deux parts composées.

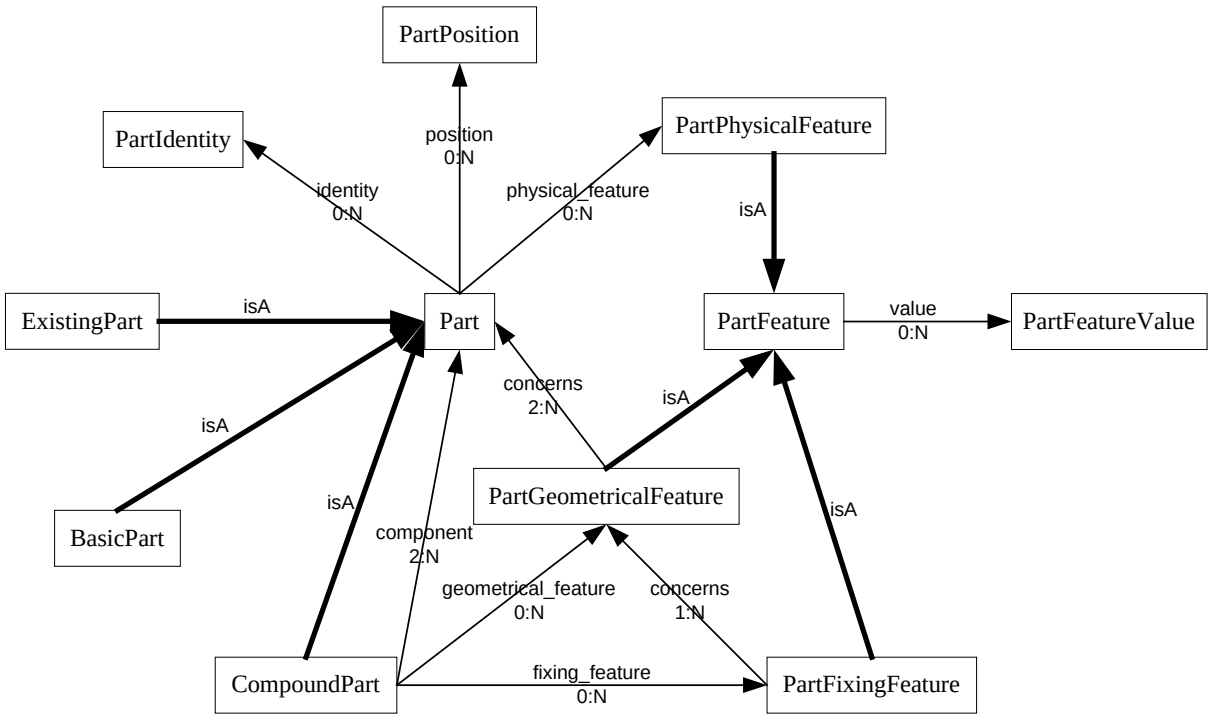


Figure 1.2 : Description graphique des classes générales du pattern *Part*

Avec l'accord de l'auteur de la thèse, nous avons modifié les cardinalités des classes générales d'attributs identités (*identity*) et positions (*position*) en faisant passer la valeur de la borne supérieure de "1" à "N" parce que, si une part possède une seule identité et une seule position, celles-ci peuvent être représentées par les instances de plusieurs classes spécifiques d'identités et de positions.

Les contraintes de cardinalité sur les attributs sont exprimées sur le graphique de la figure 1.2 sous la forme de cardinalités minimales et maximales.[6]

Les autres contraintes sont :

1. La classe générale de parts de base (*BasicPart*) et la classe générale de parts composées (*CompoundPart*) forment une partition de la classe générale de parts : une part ne peut pas être à la fois une part de base et une part composée mais elle doit être l'une des deux.
2. Aucune part n'est composée d'elle-même.
3. Les classes générales de dispositifs physiques (*PartPhysicalFeature*), géométriques (*PartGeometricalFeature*) et de fixation (*PartFixingFeature*) sont des classes exclusives : un dispositif ne peut appartenir simultanément à deux de ces classes.
4. Les dispositifs géométriques d'une part composée concernent uniquement les composants directs de cette part composée.
5. Les dispositifs de fixation d'une part composée concernent (fixent) uniquement les dispositifs géométriques de cette part composée.
6. Toutes les parts existantes possèdent des identités distinctes.
7. Aucune part ne peut être le composant de deux parts composées.

Ces contraintes sont exprimées de manière informelle. Elles sont des instances de la méta-classe *PatternConstraint*. Elles doivent être formalisées lors de la traduction du modèle obtenu par la réutilisation du pattern vers un langage spécifique.

1.3.2.2 Les méta-classes du pattern *Part*

Une classe de parts définit l'ensemble des propriétés de toutes les occurrences de part que cette classe possède. Elle peut avoir comme instances toutes les parts qui possèdent ces propriétés. Une occurrence de part peut appartenir à une ou plusieurs classes selon son état (position, composants et dispositifs comme définis dans le pattern *Part*). Elle peut aussi changer de classe lorsque des composants ou des dispositifs sont ajoutés ou enlevés. Les méta-classes de ce pattern aident à définir les classes de parts spécifiques à l'application. Ces classes de parts, qui sont les instances des méta-classes du pattern *Part*, peuvent être définies sur la base d'une des propriétés de leurs instances (position, dispositifs, état structurel, etc.) ou sur la base d'autres classes (en utilisant la notion de spécialisation avec la possibilité d'y associer des contraintes comme les contraintes de partition).

À l'instar de la distinction entre les parts de base et les parts composées, une distinction peut être faite entre les classes de parts de base (qui ne possèdent que des parts de base comme instances) et les classes de parts composées (qui ne possèdent que des parts composées comme instances).

---

[6] Une part ne doit posséder qu'une seule identité et ne peut occuper qu'une seule position mais cette identité et cette position peuvent être constituées d'un ensemble d'instances provenant chacune respectivement d'une classe spécifique d'identités ou de positions différente. Le schéma des classes générales du pattern *Part* présenté dans la thèse a donc été corrigé en ce sens : les anciennes valeurs ("0:1") des cardinalités des classes générales d'attributs identités et positions ont été remplacées pour permettre d'exprimer le caractère multiple ("0:N") des identités et des positions.

L'utilisateur peut classer les parts de différentes manières, en définissant notamment des classes de parts. Une classe de parts peut être créée pour toutes les parts qui occupent une position parmi un ensemble de positions. Avant cela, une classe de positions dont les instances sont toutes les positions acceptables (que toutes les parts de cette classe partagent sans exception) doit d'abord être définie. Une classe de parts peut aussi être définie par les ensembles des dispositifs physiques que toutes les instances de la classe doivent posséder. De manière similaire, une classe de parts peut être définie pour toutes les parts qui possèdent une identité parmi un ensemble d'identités particulières.

Pour les parts composées seulement, les dispositifs géométriques et les dispositifs de fixation peuvent aussi être utilisés : une classe de parts composées peut être caractérisée par les classes possibles des composants de ses instances et par les classes possibles des dispositifs géométriques et de fixation qui peuvent exister entre eux.

D'autres classes de parts peuvent être définies en groupant leurs instances ou en subdivisant celles-ci dans des groupes plus petits. Ce processus équivaut à définir une nouvelle classe et à placer des liens de spécialisation vers les anciennes classes. La sémantique d'une liaison de spécialisation entre les classes constitue la sémantique du sous-ensemble (incluant toutes les instances de la sous-classe dans la super classe) et est associée à l'héritage des propriétés (attributs, contraintes et usages[7]). Les classes dans une hiérarchie de spécialisation peuvent être des classes exclusives (une instance de la super classe ne peut être dans plus d'une des sous-classes en même temps), des super classes abstraites (toutes les instances de la super classe doivent être dans au moins une des sous-classes) ou des partitions (une super classe abstraite avec un ensemble de sous-classes exclusives).

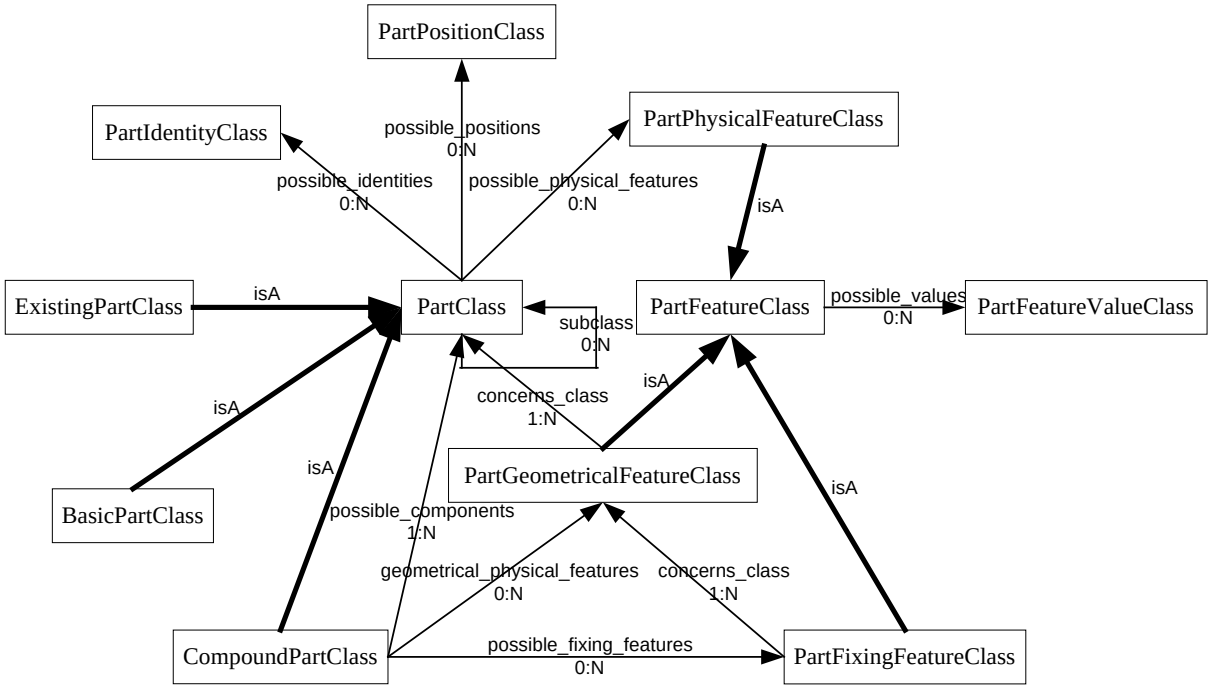


Figure 1.3 : Description graphique des méta-classes du pattern *Part*

[7] Les usages font référence aux actions qui peuvent s'exercer sur les parts. Par exemple, une part composée peut être décomposée en ses parts composantes. Les usages sont des propriétés héritées parce que tout ce que les instances d'une classe de parts peuvent faire, les instances des sous-classes de cette classe de parts peuvent également le faire.

Les contraintes de cardinalité sur les classes d'attributs sont exprimées sur le graphique de la figure 1.3 sous la forme de cardinalités minimales et maximales.

Les autres contraintes sont :

1. La méta-classe de parts de base (*BasicPartClass*) et la méta-classe de parts composées (*CompoundPartClass*) sont des méta-classes exclusives. Cela signifie qu'aucune classe de parts ne peut être à la fois une classe de parts de base et une classe de parts composées.
2. Les méta-classes de dispositifs géométriques (*PartGeometricalFeatureClass*), physiques (*PartPhysicalFeatureClass*) et de fixation (*PartFixingFeatureClass*) sont des méta-classes exclusives. Cela signifie, par exemple, qu'une classe de dispositifs physiques ne peut être ni une classe de dispositifs géométriques ni une classe de dispositifs de fixation. En d'autres termes, les dispositifs physique, géométrique et de fixation sont des concepts distincts.
3. Si une occurrence de la méta-classe de parts composées (*CompoundPartClass*) est définie sur la base d'une (ou de plusieurs) occurrences de la méta-classe de parts (*PartClass*) – comme ses composants possibles – et d'une (ou de plusieurs) occurrences de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*), alors ces occurrences de la méta-classe de dispositifs géométriques doivent être définies sur la base d'un sous-ensemble des instances de la méta-classe de parts qui sont utilisées (comme composants possibles) pour définir cette occurrence de la méta-classe de parts composées.
4. Si une occurrence de la méta-classe de parts composées (*CompoundPartClass*) est définie sur la base d'une (ou de plusieurs) occurrences de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*) et d'une (ou de plusieurs) occurrences de la méta-classe de dispositifs de fixation (*PartFixingFeatureClass*), alors ces occurrences de la méta-classe de dispositifs de fixation doivent être définies sur la base d'un sous-ensemble des occurrences de la méta-classe de dispositifs géométriques qui sont utilisées pour définir cette occurrence de la méta-classe de parts composées.
5. Il existe tout au plus une instance de chaque méta-classe d'attributs parmi deux instances de méta-classe. Par exemple, une classe de parts ne peut pas être définie en l'attachant deux fois (donc en utilisant deux fois la même classe d'attributs) à ses composants possibles. Une seule instance de la méta-classe d'attributs doit être employée et celle-ci doit utiliser les spécifications des cardinalités appropriées.
6. Les instances de la méta-classe d'attributs sous-classes (*subclass*) forment un ordre partiel sur les classes de parts. Cette contrainte implique la hiérarchisation de toutes les sous-classes de parts.

1.3.2.3 Le lien entre les classes générales et les méta-classes du pattern *Part*

Le pattern *Part* combine les méta-classes et les classes générales : une classe générale (respectivement un classe générale d'attributs) est définie pour chaque méta-classe (respectivement pour chaque méta-classe d'attributs) du domaine d'application. Chaque classe générale définit les éléments communs à toutes les instances d'une méta-classe spécifique définie au niveau du pattern. Le même raisonnement est valable pour les classes d'attributs. Chaque instance de la méta-classe est automatiquement interprétée comme une spécialisation de la classe générale et hérite par conséquent de toutes ses propriétés (dont les attributs possibles et obligatoires, ainsi que les contraintes et les usages possibles). En effet, comme c'est habituellement conseillé dans la modélisation conceptuelle et les approches orientées objets, les éléments communs doivent être extraits d'une seule classe et hérités par les sous-classes sans être redéfinis chaque fois dans ces sous-classes.

Le premier lien consiste à définir chaque classe générale (et classe générale d'attributs) comme une instance de la méta-classe correspondante (respectivement de la méta-classe d'attributs correspondante) :

- la classe générale de parts (*Part*) est une instance de la méta-classe de parts (*PartClass*);
- la classe générale de parts de base (*BasicPart*) est une instance de la méta-classe de parts de base (*BasicPartClass*);
- la classe générale de parts composées (*CompoundPart*) est une instance de la méta-classe de parts composées (*CompoundPartClass*);
- … et de manière similaire avec la classe générale de positions (*PartPosition*), la classe générale d'identités (*PartIdentity*), la classe générale de dispositifs (*PartFeature*), … et les méta-classes correspondantes, respectivement la méta-classe de positions (*PartPositionClass*), la méta-classe d'identités (*PartIdentityClass*), la méta-classe de dispositifs (*PartFeatureClass*), …;
- la classe générale d'attributs positions (*position*) qui lie la classe générale de parts (*Part*) et la classe générale de positions (*PartPosition*) est une instance de la méta-classe d'attributs positions (*possible_positions*) qui unit la méta-classe de parts (*PartClass*) à la méta-classe de positions (*PartPositionClass*).
- … et de manière similaire avec la classe générale d'attributs identités (*identity*), la classe générale d'attributs dispositifs (*feature*), … et les méta-classes d'attributs correspondantes, respectivement la méta-classe d'attributs identités (*possible_identities*), la méta-classe d'attributs dispositifs (*possible_features*), …;

Le second lien consiste à définir chaque instance d'une méta-classe (et d'une méta-classe d'attributs) comme une spécialisation de la classe générale correspondante (respectivement de la classe générale d'attributs correspondante) :

- chaque instance d'une méta-classe de parts (*PartClass*) est une spécialisation de la classe générale de parts (*Part*). Ceci est intuitivement correct parce que les instances de l'instance d'une méta-classe de parts sont des parts. Celles-ci doivent évidemment respecter les contraintes sur les propriétés de toutes les parts définies dans la classe générale de parts;
- chaque instance de la méta-classe de parts de base (*BasicPartClass*) est une spécialisation de la classe générale de parts de base (*BasicPart*), c'est-à-dire que chaque instance d'une instance de la méta-classe de parts de base est une instance de la classe générale de parts de base;
- chaque instance de la méta-classe de parts composées (*CompoundPartClass*) est une spécialisation de la classe générale de parts composées (*CompoundPart*), c'est-à-dire que chaque instance d'une instance de la méta-classe de parts composées est une instance de la classe générale de parts composées.
- … et de manière similaire avec la méta-classe de positions (*PartPositionClass*), la méta-classe d'identités (*PartIdentityClass*), la méta-classe de dispositifs (*PartFeatureClass*), … et la classe générale correspondante, respectivement la classe générale de positions (*PartPosition*), la classe générale d'identités (*PartIdentity*), la classe générale de dispositifs (*PartFeature*), …
- Chaque instance de la méta-classe d'attributs positions (*possible_positions*) qui lie la méta-classe de parts (*PartClass*) et la méta-classe de positions (*PartPositionClass*) est une spécialisation de la classe générale d'attributs positions (*position*) qui unit la classe générale de parts (*Part*) à la classe générale de positions (*PartPosition*).
- … et de manière similaire avec la méta-classe d'attributs identités (*possible_identities*), la méta-classe d'attributs dispositifs (*possible_features*), … et la classe générale d'attributs correspondante, respectivement la classe générale d'attributs identités (*identity*), la classe générale d'attributs dispositifs (*feature*), …

Ces règles sont importantes parce qu'elles garantissent que si une classe spécifique d'application est définie (par exemple *Boulon*, la classe des boulons), toutes les contraintes qui concernent les instances de la classe générale de parts (*Part*) s'appliquent automatiquement à l'instance de *Boulon* parce que *Boulon* est une spécialisation de la classe générale de parts et hérite par conséquent de ses contraintes. Ceci est également vrai pour les contraintes sur les instances des méta-classes d'attributs puisqu'elles sont définies comme les spécialisations des classes générales d'attributs du pattern *Part* auxquelles sont associées les contraintes liées à ce pattern (comme les contraintes de cardinalité).

Les liens entre les méta-classes et les classes générales impliquent aussi d'autres contraintes. En voici un exemple : chaque instance de la méta-classe de parts composées (*CompoundPartClass*) est exclusive vis-à-vis des instances de la méta-classe de parts de base (*BasicPartClass*) parce qu'elles sont toutes des spécialisations respectives de la classe générale de parts composées (*CompoundPart*) et de la classe générale de parts de base (*BasicPart*) qui sont des classes exclusives. Cela signifie que les instances d'une classe de parts composées ne peuvent pas être membres d'une classe de parts de base et vice-versa.

1.3.2.4 Règles de réutilisation du pattern *Part*

Le pattern *Part* est réutilisé et adapté au moyen d'une combinaison des trois méthodes suivantes :

- par simple copie d'une des classes générales;
- par instanciation des méta-classes et des méta-classes d'attributs (définissant ici les classes et les classes d'attributs spécifiques à l'application);
- par substitution de paramètres.

Première méthode : la réutilisation par copier/coller. Les classes générales (et les classes générales d'attributs) peuvent être utilisées comme des classes régulières dans une application particulière si les dispositifs qui sont définis dans cette application traduisent exactement ceux définis dans le pattern. La manière la plus simple de réutiliser une classe générale consiste à l'importer (la copier) dans le modèle. La nouvelle classe définit alors des caractéristiques universelles qui s'appliquent à tous les états, parts ou attributs de l'application. Cette méthode peut rendre obligatoire la réutilisation d'autres classes du pattern si la classe copiée est liée à ces classes au moyen d'un attribut obligatoire. Pour toute classe générale (et classe générale d'attributs) réutilisée, la méta-classe (méta-classe d'attributs) correspondante doit être copiée dans le modèle. Chaque classe générale doit être insérée une seule fois dans le modèle parce qu'elle représente les caractéristiques communes à tous les objets d'une classe donnée. Par exemple, *CompoundPart* doit être la seule classe à posséder comme instances toutes les occurrences de part composée du système. Les sous-classes des classes générales ne doivent pas être définies directement : elles doivent l'être si possible par instanciation des méta-classes comme expliqué ci-dessous.

Deuxième méthode : la réutilisation par instanciation. Le pattern *Part* peut être réutilisé en définissant des instances de ses méta-classes. Ceci permet d'élaborer des classes spécifiques à l'application. Conformément à la définition des méta-classes et des classes générales énoncée dans les sections précédentes, cette instanciation implique que des liens de spécialisation sont créés depuis les instances jusqu'aux classes générales du pattern. Par exemple, chaque instance de la méta-classe de parts composées (*CompoundPartClass*) devient une spécialisation de la classe générale de parts composées (*CompoundPart*). Par conséquent, chaque classe spécifique de parts composées est une sous-classe de la classe

générale de parts composées. Grâce à ce procédé, les nouvelles classes ou classes d'attributs héritent automatiquement de toutes les propriétés de la classe générale ou de la classe générale d'attributs correspondante. Ces propriétés, qui incluent les attributs, les contraintes et les usages, ne doivent donc pas être redéfinies explicitement.

Troisième méthode : la substitution d'un paramètre. Toutes les classes générales du pattern *Part* sont considérées comme des paramètres qui peuvent être substitués. L'objectif d'une substitution est de remplacer une classe générale par une autre classe définie par l'utilisateur qui possède des attributs ou des contraintes supplémentaires qui s'appliquent à toutes les instances de la classe d'objets de l'application. Toutefois, cette classe spécifique doit posséder au moins les mêmes dispositifs (notamment les attributs, les contraintes et les usages) que la classe générale qu'elle remplace afin de préserver la sémantique de la classe générale. Cette paramétrisation est exprimée en présentant la substitution (la classe définie par l'utilisateur) comme une sous-classe du paramètre (la classe générale remplacée). Cette spécialisation permet de faire valoir que les attributs, les contraintes et les usages qui s'appliquent au paramètre s'appliquent aussi à la substitution puisque chaque instance de la sous-classe (la substitution) est aussi une instance de la super classe (le paramètre). Cependant, le paramètre substitué doit devenir une classe abstraite : les seules instances qu'il peut avoir doivent être des instances de la substitution. Cette contrainte permet d'exprimer que toutes les instances du paramètre possèdent les propriétés qui sont définies dans la classe de substitution. Au sein du modèle, elle se présente sous la forme d'un lien de spécialisation entre le paramètre et sa substitution, un lien qui est défini comme une partition, ce qui signifie qu'une instance du paramètre est aussi une instance de la substitution et vice-versa.

1.3.2.5 Les classes spéciales de parts composées

Les classes spéciales de parts composées qui interviennent dans la construction des tampons et des piles de parts sont définies par instanciation des méta-classes du pattern *Part*. Comme elles se rencontrent souvent dans les systèmes manufacturiers, elles ont été inclues dans le pattern avec l'intention de pouvoir les réutiliser par instanciation de la méta-classe d'attributs sous-classes (*subclass*).

Les parts existantes sont toujours présentes quelque part dans le système modélisé. Elles sont habituellement placées dans des tampons dans l'attente d'être utilisées. Lorsqu'elles sont traitées, elles se trouvent également dans les tampons des machines. Ainsi, elles sont généralement transférées de tampons (de sortie) en tampons (d'entrée). Le tampon, qui constitue le premier type de parts composées spéciales, peut donc être défini comme un endroit où un certain nombre de parts peuvent demeurer quelque temps.

Le tampon vide (*EmptyBuffer*) est une part de base spéciale qui est sensée être composée d'autres parts qui peuvent constituer son contenu. Contrairement aux parts habituelles, les tampons possèdent une position fixe. Cependant, il est intéressant de les identifier et de les modéliser séparément.

Le tampon rempli (*FilledBuffer*) est une part composée dont le tampon vide est un composant (directement ou comme composant intérieur d'un des composants du tampon rempli). Les autres composants d'un tampon rempli sont appelés ses "contenus".

Les contraintes de cardinalité sur les attributs sont exprimées sur le graphique de la figure 1.4 sous la forme de cardinalités minimales et maximales.[8]

[8] La classe spéciale des tampons remplis (*FilledBuffer*) est une spécialisation de la classe générale des parts composées (*CompoundPart*). Chacune de ses instances doit donc posséder au moins deux composants. Le

Les autres contraintes sont :

1. Un tampon vide (*EmptyBuffer*) est une part existante (*ExistingPart*).
2. Un tampon vide (*EmptyBuffer*) possède une position fixe (*PartPosition*).
3. Chaque composant d'un tampon rempli (*FilledBuffer*) est une part existante. Mais l'inverse n'est pas vrai : des parts peuvent exister sans être dans un tampon. C'est le cas, par exemple, d'une part au moment où elle est transférée entre deux tampons.

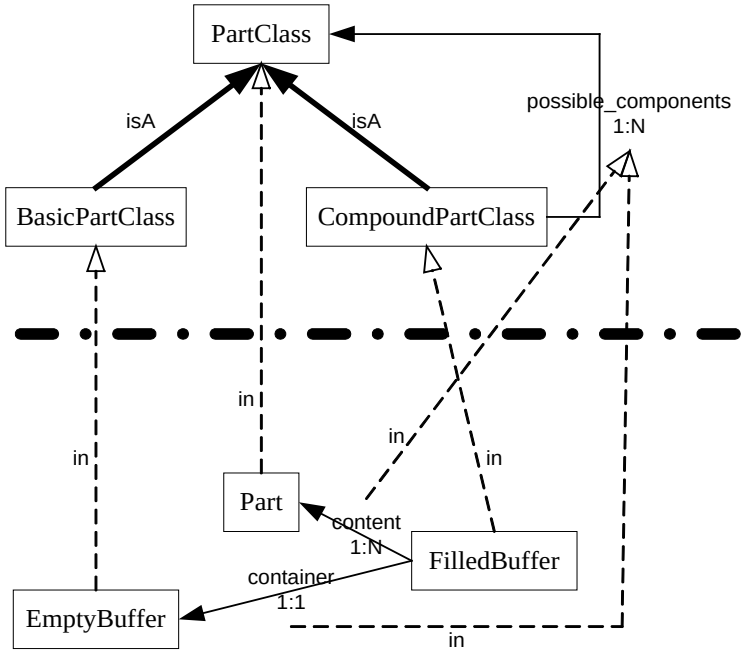


Figure 1.4 : Description graphique de la classe des tampons du pattern *Part*

De manière similaire aux règles de réutilisation exposées dans la section précédente, les classes spécifiques de tampons peuvent être définies comme des spécialisations de la classe de tampons remplis (*FilledBuffer*) en ajoutant une instance de la méta-classe d'attributs sous-classes (*subclass*) entre chacune d'elles et cette classe. Elles peuvent aussi être définies comme les nouvelles instances de la méta-classe de parts composées (*CompoundPartClass*). Mais, dans ce cas, toutes les contraintes se rapportant aux tampons doivent être associées à cette nouvelle instance. Ces deux méthodes de réutilisation permettent notamment de définir une classe de tampons sur la base des types de contenus qu'elle peut accepter (des tampons de boulons par exemple).

Des classes spéciales de tampons peuvent également être définies comme des classes de tampons de sortie (tampons situés à la fin d'un processus de production) par spécialisation des classes de tampons vides (*EmptyBuffer*).

---

schéma de la classe des tampons du pattern *Part* présenté dans la thèse a été corrigé en ce sens : la valeur de la borne inférieure de la cardinalité de la classe spéciale d'attributs contenus (*content*) est passée de zéro à un.

Les piles de parts sont des parts composées spéciales dont les composants sont disposés dans des configurations particulières, c'est-à-dire qu'ils sont "pilés" (placés) les uns sur les autres. Le pattern *Part* est utilisé pour définir la classe des piles de parts (*PileOfParts*) dans un système manufacturier : la génération (d'occurrences) de classes spéciales de dispositifs géométriques permet d'exprimer cette notion de "pilés".

Les classes suivantes ont été définies par instanciation de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*) :

- la classe de contiguïté (*PartContiguity*) décrit qu'un ensemble de parts sont si près les unes des autres qu'il n'est pas possible d'ajouter un composant entre ces parts;
- la classe d'enclos (*PartEnclosure*) décrit qu'un ensemble de parts empêchent collectivement l'accès à un ensemble d'autres parts. Les parts dont l'accès est rendu impossible sont dites encloses (*enclosed*) par les autres parts qui sont appelées des parts enclosant (*enclosing*);
- la classe de contenants (*PartContainment*) décrit qu'un ensemble de parts sont contenues dans une seule part (mais pas nécessairement encloses par cette part). La part contenant est appelée un conteneur (*container*). Chaque part contenue est désignée comme un contenu (*content*).

Certaines opérations sur une part composée ne peuvent pas être exécutées lorsque celle-ci contient des dispositifs géométriques spéciaux parmi ses composants. Par exemple, un composant ne peut pas être enlevé d'un composé s'il est enclos par d'autres composants. De la même manière, il n'est pas possible d'ajouter une part à un composé de telle sorte qu'elle soit contiguë à un composant enclos ou qu'elle soit simultanément contiguë à deux composants qui sont déjà contigus. Les trois classes spécifiques de dispositifs géométriques décrites ci-dessus correspondent à des contraintes d'accessibilité que l'on retrouve habituellement dans un processus de planification. Elles font donc référence à la possibilité physique ou non de réaliser certaines opérations sur les parts.

Ces trois nouvelles classes permettent de définir les classes spéciales de parts composées. Dans une pile de parts, chaque composant est contigu au composant suivant et aucun composant n'est contigu à un composant situé plus loin dans la pile. En conséquence, on peut seulement ajouter des composants à l'une des deux extrémités de la pile mais on peut toujours enlever un composant au milieu de la pile. Une classe de piles de parts (*PileOfParts*) est alors définie comme une instance de la méta-classe de parts composées (*CompoundPartClass*) qui contient comme composants des instances de la méta-classe de parts (*PartClass*) et qui possède un certain nombre de dispositifs géométriques dérivés de la classe de contiguïté (*PartContiguity*).

Une pile est généralement mise dans un conteneur. Le conteneur est contigu à toutes les parts de la pile et les enclos de telle manière qu'il est impossible d'en enlever une seule, à l'exception de celles situées aux deux extrémités. Cette configuration définit la classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*).

Si le conteneur est fermé à une extrémité de la pile, la part située à l'autre extrémité est la seule accessible. Dans ce type de classes de conteneurs (*FIFOPileOfPartsInContainer*) qui est une spécialisation de la classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*), la part située à l'extrémité fermée de la pile est enclose par le conteneur.

Le graphe de la figure 1.5 montre les classes nécessaires à la définition des piles de parts. La partie supérieure du graphe délimitée par l'épaisse ligne pointillée horizontale correspond aux méta-classes du pattern *Part* tandis que la partie inférieure correspond aux classes spéciales de parts composées et de dispositifs géométriques. Les flèches en pointillés caractérisent les liaisons d'instanciation entre les deux parties. La classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*) est partiellement définie sur le graphe.

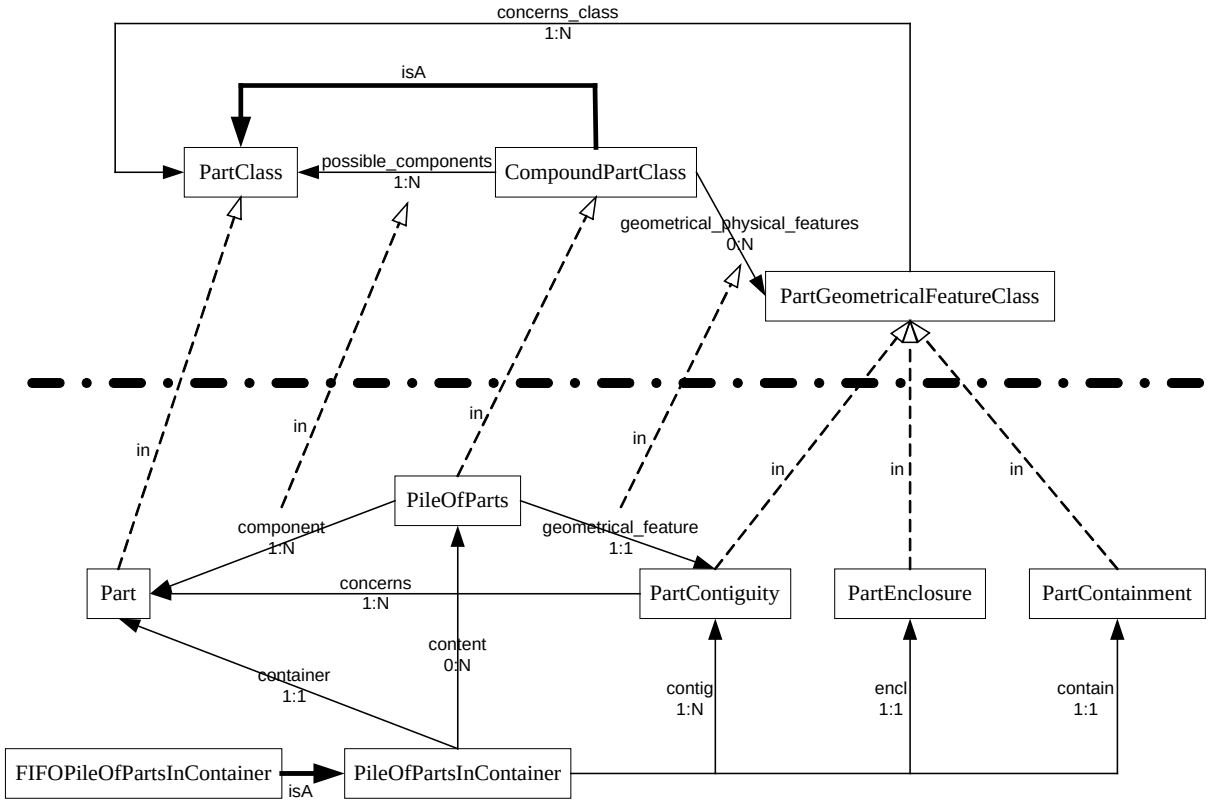


Figure 1.5 : Description graphique de la classe des piles de parts (*PileOfParts*) du pattern *Part*

Cependant, quelques erreurs se sont glissées dans la description de la classe des piles de parts. D'autre part, le modèle lui-même ne satisfait pas à exprimer toutes les subtilités liées aux classes spéciales de parts composées qui interviennent dans la construction des piles. Par conséquent, nous avons remanié la définition des piles en concertation avec l'auteur de la thèse et nous avons choisi de la présenter en nous plaçant du point de vue de la classe d'une pile de parts dans un conteneur en la définissant, cette fois, de manière exhaustive.

La classe d'une pile de parts dans un conteneur est modélisée sur le même schéma que celui de la classe des tampons, c'est-à-dire au moyen d'une classe qui contient les parts spéciales vides et qui est associée à la classe principale. Pour définir la classe d'une pile de parts dans un conteneur, nous avons donc besoin de créer une nouvelle classe qui est la classe des conteneurs vides (*EmptyContainer*). Un conteneur vide est une part de base spéciale qui est sensée être composée de piles (une seule pile en réalité comme nous le verrons plus loin) qui peuvent constituer son contenu. Nous considérons alors la classe d'une pile de parts dans un conteneur comme une spécialisation de la classe des conteneurs remplis (*FilledContainer*). Le conteneur rempli devient une part composée dont le conteneur vide est un composant obligatoire.

Lorsqu'une pile n'est ouverte qu'à une seule extrémité, l'unique part qui peut être enlevée est soit celle qui a été placée la première dans la pile (First In First Out ou FIFO), soit celle qui a été placée la dernière (Last In First Out ou LIFO). Cette pile est une instance d'une des deux classes d'une pile de parts dans un conteneur fermé à une extrémité (*FIFO-* ou *LIFO-PileOfPartsInContainer*) qui sont des spécialisations de la classe d'une pile de parts dans un conteneur.

Afin de respecter la définition de la classe générale d'attributs composants qui associe ici la classe de piles à la classe de parts, nous avons modifié sa cardinalité en lui attribuant la valeur "2:N". Nous partons aussi du principe que l'utilisateur emploie une pile plutôt qu'une part composée "standard" pour exprimer que des parts sont contiguës. Or, une part seule n'est jamais contiguë. Cette nouvelle valeur permet en outre de vérifier la première contrainte relative aux classes spéciales (voir ci-dessous) qui stipule que chaque composant d'une pile est contigu à deux autres composants, à l'exception des deux composants situés aux extrémités qui sont contigus à un seul composant. En effet, si une pile pouvait ne comprendre qu'une seule part, celle-ci ne serait contiguë à aucune autre part.

Si un conteneur est une part composée, par définition, il doit être constitué d'au moins deux composants. Puisqu'il contient toujours un et un seul conteneur vide, il doit également contenir au moins une pile. La borne inférieure de la cardinalité de la classe spéciale d'attributs contenus (*content*) doit donc être équivalente à un. D'autre part, nous ne percevons pas l'intérêt pour un conteneur de renfermer plusieurs piles car ce dernier sert uniquement à définir les éléments d'une pile du point de vue de leur contiguïté, de leur enclosure et de leur contenance. La borne supérieure de la même cardinalité doit donc aussi être égale à un.

La manière dont est modélisée la contiguïté pose également des difficultés car si une pile ne possède qu'un seul dispositif de contiguïté, comment exprimer la contiguïté entre deux parts spécifiques ? Comment traduire l'ordre d'empilement des parts ? Et sans le conteneur, comment désigner les parts situées aux extrémités de la pile et qui ne sont contiguës qu'à une seule part ? Pour résoudre ces problèmes, nous postulons que les éléments d'une pile, qui sont placés les uns derrière les autres, sont contigus deux à deux et que le caractère adjacent de deux parts est exprimé par le dispositif de contiguïté. La cardinalité de la classe spéciale d'attributs concernés (*concerns*) qui associe la classe spéciale de contiguïté (*PartContiguity*) à la classe générale de parts (*Part*) est modifiée pour prendre la valeur "2:2". Une pile peut donc posséder plusieurs dispositifs de contiguïté dont un obligatoire car elle contient toujours au moins deux parts. Par ailleurs, comme indiqué ci-dessus, la classe de piles est censée disposer "d'un certain nombre" de dispositifs géométriques dérivés de la classe de contiguïté. En conséquence, la cardinalité de la classe d'attributs dispositifs géométriques (*geometrical_feature*) qui associe la classe de piles à la classe de contiguïté est fixée à "1:N". Parallèlement, nous avons assigné la même valeur à la cardinalité de la classe spéciale d'attributs contiguïtés (*contig*).

Le graphe de la figure 1.6 montre les classes nécessaires à la définition d'une pile de parts dans un conteneur. La partie supérieure du graphe délimitée par l'épaisse ligne pointillée horizontale correspond aux méta-classes du pattern *Part* tandis que la partie inférieure correspond aux classes spéciales de parts et de dispositifs géométriques. Les flèches en pointillés caractérisent les liaisons d'instanciation entre les deux parties. Les classes d'attributs concernés (*concerns*) qui associent les classes spéciales de dispositifs géométriques à la classe générale de parts font autant référence à la définition de la classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*) qu'à celle de la classe des piles de parts (*PileOfParts*). La borne inférieure de la cardinalité de la classe d'attributs concernés dont l'origine est la classe spéciale d'enclos (*PartEnclosure*) est nulle parce que si une pile ouverte aux deux extrémités ne contient que deux parts, celles-ci ne sont pas encloses. Par contre, la borne inférieure de la cardinalité de la classe d'attributs concernés dont l'origine est la classe de contenants (*PartContainment*) est égale à deux car une pile possède toujours au moins deux parts.

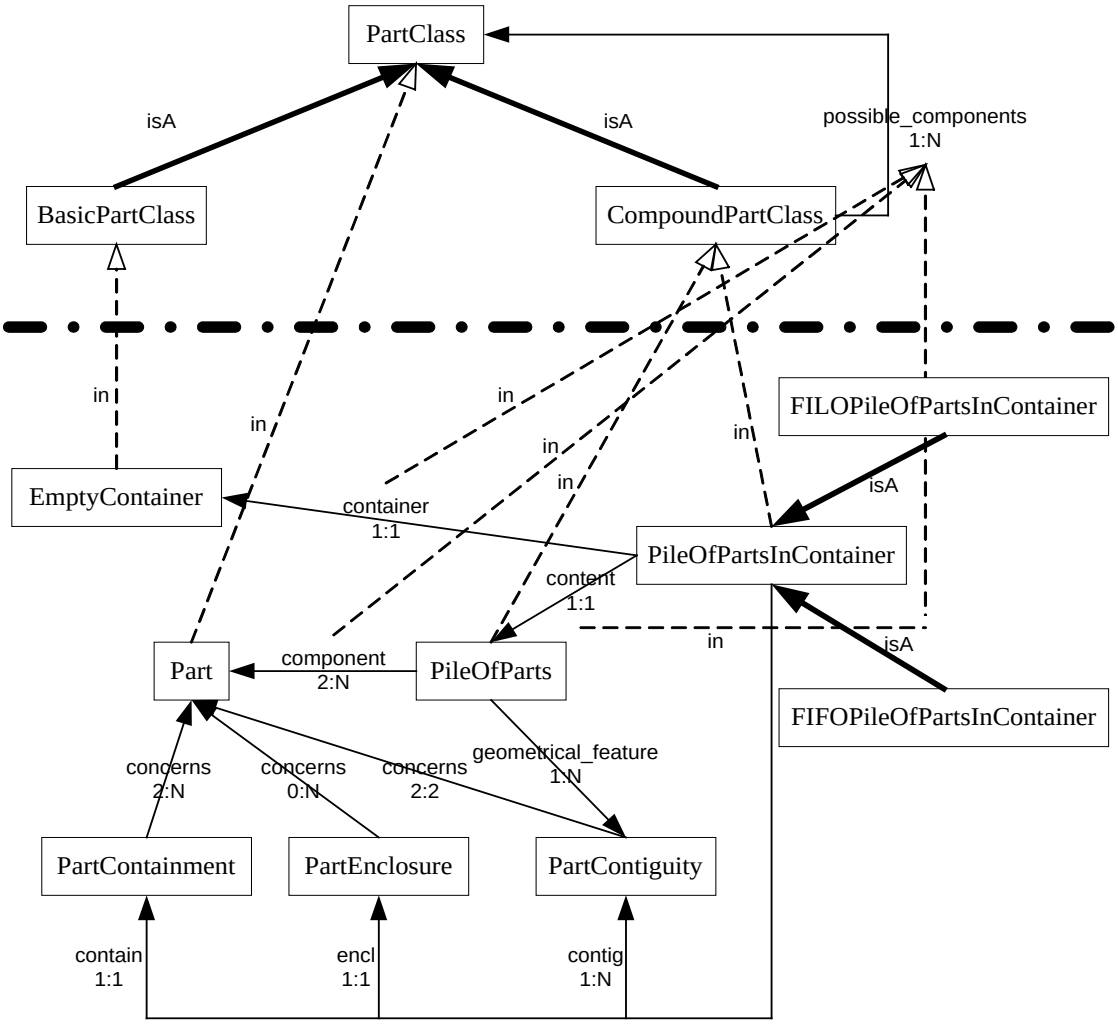


Figure 1.6 : Description graphique de la classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*) du pattern *Part*

Les contraintes de cardinalité sur les attributs sont exprimées sur le graphique de la figure 1.6 sous la forme de cardinalités minimales et maximales.

Les autres contraintes sont :

1. Dans une classe de piles de parts (*PileOfParts*), chaque composant est contigu à exactement deux autres composants, à l'exception de deux d'entre eux (localisés aux extrémités) qui sont contigus à un seul composant.
2. Dans la classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*), un conteneur est contigu à toutes les parts de la pile et les enclos de telle manière qu'il est impossible d'enlever l'une d'entre elles, à l'exception de celles situées aux deux extrémités.
3. Dans la classe d'une pile de parts dans un conteneur fermé à une extrémité (*FIFOPile-OfPartsInContainer* ou *LIFOPileOfPartsInContainer*), le conteneur enclos une part à l'une des extrémités.

Des classes de tampons peuvent être déclarées comme des spécialisations des classes de piles de parts. Ce principe permet la définition de classes de tampons LIFO.

Chapitre 2 :

Présentation du logiciel Microsoft Visio

Afin de construire des modèles dessinés de "parts", sur base du pattern *Part* présenté au chapitre précédent, notre choix s'est porté sur l'éditeur graphique Visio développé par la firme Microsoft. Nous commencerons donc par une présentation générale de ce logiciel. Nous invoquerons ensuite rapidement les raisons qui nous ont poussé à choisir ce produit. Dans la dernière partie, nous passerons en revue les différents concepts de Visio qui seront utiles à notre travail.

2.1. Présentation générale du logiciel Visio

Microsoft Office Visio est un éditeur graphique qui permet de créer une grande variété de diagrammes pour lesquels l'utilisateur bénéficie de bibliothèques métiers : diagrammes organisationnels (diagrammes de flux, organigrammes, barres de planning, diagrammes de blocs, matrices commerciales et marketing etc.), schémas techniques (agencements d'espace, schémas mécaniques, électriques, électroniques, pneumatiques, hydrauliques etc.) et schémas informatiques (réseaux, services d'annuaire, modélisations de bases de données, sites Web, modélisations de logiciels, etc.). L'objectif des concepteurs est d'aider les entreprises et les professionnels à documenter et à partager visuellement des idées, des processus et des informations techniques afin de faciliter la compréhension de leurs interlocuteurs et de renforcer leur argumentation auprès d'eux.

Visio est disponible en deux versions. La version standard propose essentiellement des diagrammes pour la présentation d'informations et est destinée à des responsables administratifs et financiers, à des responsables des ressources humaines, à des chefs de projet et à des responsables de la vente et du marketing. La version professionnelle permet en plus de visualiser des systèmes existants et d'en concevoir de nouveaux à l'aide de diagrammes. Elle s'adresse plutôt à des utilisateurs évoluant dans le domaine de l'industrie et des technologies (technologies de l'information, développement et ingénierie).

Visio est conçu pour interagir avec différents types de bases de données, notamment par la création automatique de diagrammes ou par l'extraction de données des schémas sous la forme de code XML, Excel, Word, SQL Server ou autre. Visio permet aussi de créer des pages Web et fournit une interface utilisateur de type navigateur pour l'accès aux données présentées sur le dessin. La surface de dessin peut également être intégrée et programmée dans des logiciels métiers différents de Visio.

D'autre part, un kit de développement permet de créer des applications pour la plate-forme Visio. Bien documenté, il inclut de nombreux exemples et outils. Il fournit un ensemble de fonctions, de classes et de procédures réutilisables pour la plupart des tâches courantes de développement. Il prend en charge une large gamme de langages, dont Visual Basic, Visual Basic .NET, Visual C# .NET et Visual C++. Il propose également des exemples d'applications qui illustrent l'automatisation et l'intégration de Visio avec les autres produits de Microsoft. [Visio 2003]

2.2. Le choix du logiciel Visio

La préférence accordée au logiciel Visio se justifie par des raisons tant subjectives qu'objectives.

Les raisons subjectives d'abord. Visio est doté d'une présentation qui allie simplicité et clarté. En outre, la manipulation des outils et la navigation dans les menus sont relativement faciles et intuitives, ce qui facilite une prise en main rapide du logiciel. Comme la plupart des produits Microsoft, il a bénéficié d'une attention particulière au niveau du design et de l'ergonomie. Tous ces facteurs ont sans doute permis de nous sentir directement à l'aise avec lui.

D'autre part, nous étions déjà familiarisé avec Visio : nous l'avions employé pour élaborer des schémas dans le cadre du Laboratoire de Méthodologie de Développement de Logiciel dirigé par Naji Habra en 2003-2004.

Les raisons objectives ensuite. Visio fait partie de la famille Office dont il bénéficie des fonctionnalités telles que les raccourcis clavier, les menus personnalisés, la correction automatique, le vérificateur orthographique, le système d'aide, etc. Par ailleurs, l'interface familière et les nombreuses fonctionnalités communes à cette gamme de produits permettent d'être rapidement productif en tirant profit des techniques mises en pratique avec l'un ou l'autre de ces logiciels. Visio s'intègre aussi facilement avec les autres applications Office : ses diagrammes peuvent être insérés puis mis à jour dans des documents Word, des présentations PowerPoint ou des courriers électroniques.

Afin de vérifier les contraintes liées au pattern *Part* et, éventuellement, de traduire en langage AlbertII les modèles graphiques construits, nous avions besoin d'un outil de programmation permettant de construire des solutions complexes. Or, Visio offre un kit de développement adapté à la plate-forme Visio capable de concevoir ce type d'application. De plus, il fournit une documentation étendue avec de nombreux exemples pour simplifier et accélérer le développement d'applications personnalisées.

Visio propose également des formes personnalisables, un modèle d'objet souple, un format de fichier XML et d'autres fonctionnalités pour créer des solutions visuelles personnalisées et réutilisables. Des données importantes peuvent ainsi être stockées dans les propriétés personnalisées des formes afin d'être exportées pour être analysées dans d'autres programmes. Elles peuvent aussi être utilisées pour générer des rapports dans des formats texte, Excel, HTML et XML. Le nouveau format de fichier XML offre en outre une interopérabilité avec les autres applications qui le prennent en charge et facilite le stockage et l'échange d'informations contenant des diagrammes, notamment des données qui ne sont associées ni à une page ni à une forme. Or, les modèles graphiques de "parts" contiennent de nombreuses données sur les classes : nom de la classe, nom de la classe héritée (méta-classe), numéro de la classe, cardinalité de la relation et types de valeurs). Ces données doivent pouvoir être analysées et, éventuellement, exportées vers d'autres formats pour être manipulées.

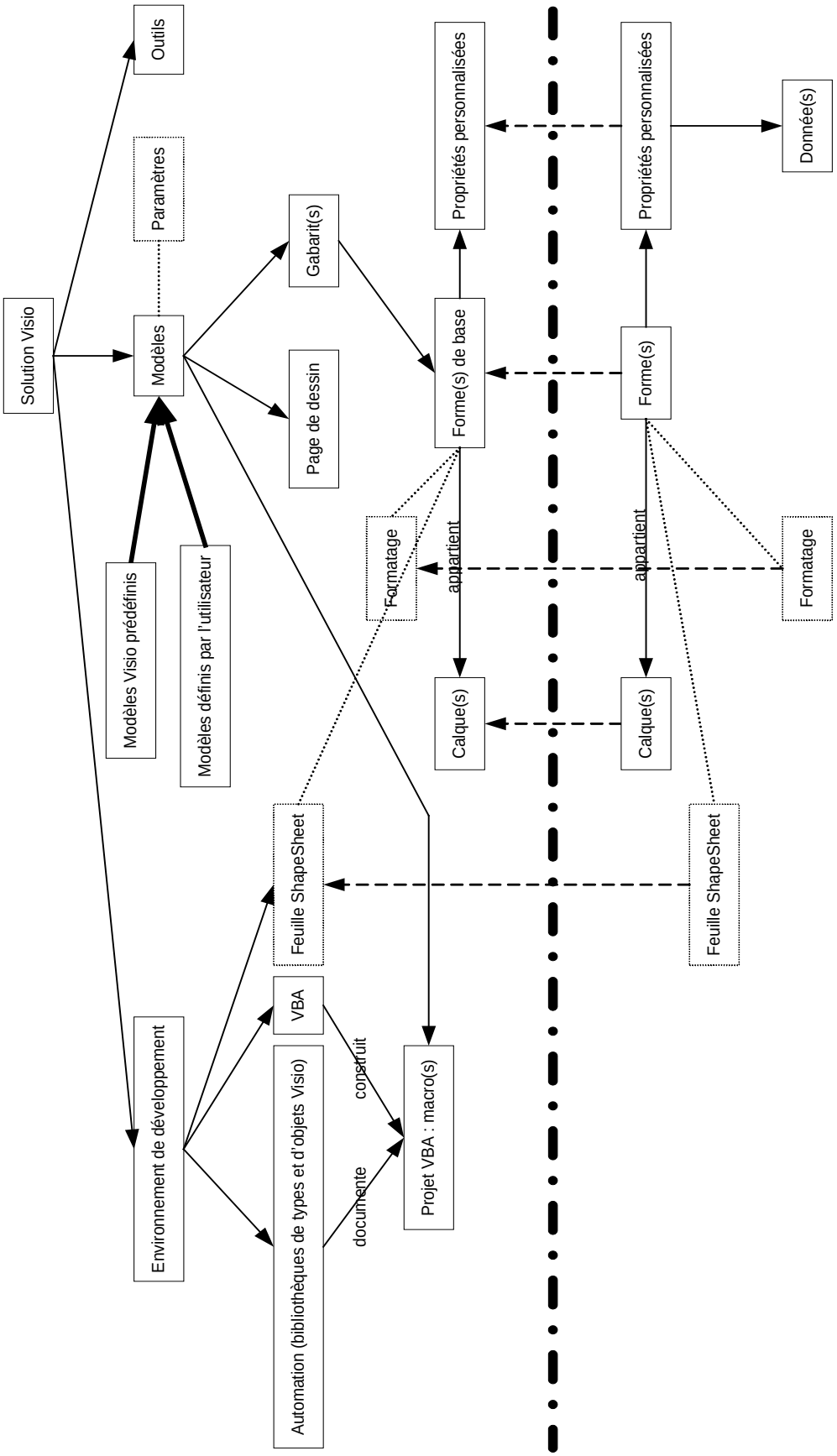


Figure 2.1 : Graphe des concepts Visio

2.3. Quelques concepts Visio

La présentation des concepts est basée sur la version logicielle de Visio utilisée dans le cadre du mémoire : Microsoft Visio Professional 2002 qui la première version de Visio développée par Microsoft. [Visio 2002]

Le graphe de la figure 2.1 illustre les liens entre les différents concepts de Visio. Les concepts spécifiques, qui représentent des objets concrets, sont encadrés par un trait plein tandis que les propriétés qui les définissent sont entourées par des pointillés. Les flèches épaisses symbolisent un lien de spécialisation qui se présente sous la forme d'une partition. Les flèches avec un trait plein traduisent une relation de composition. Lorsque ce n'est pas le cas, un attribut spécifie le type de relation. Les flèches en pointillés caractérisent une liaison d'instanciation. Les traits pointillés associent les concepts spécifiques à leurs propriétés. Le graphe se limite à présenter les concepts Visio utilisés dans notre travail. La partie supérieure du graphe (délimitée par l'épaisse ligne pointillée horizontale) correspond plutôt à l'espace de travail (l'interface et les outils définis par Visio ou l'utilisateur) tandis que la partie inférieure fait davantage référence au dessin créé par l'utilisateur. Dans ce cas, on parle aussi de projet, terme conceptuel spécifique à Visio, ou de document, expression plus technique désignant un ouvrage autonome créé dans une application et pourvu d'un nom de fichier univoque.

L'architecture globale du système, appelée solution Visio, est une combinaison de formes et de programmes qui représentent le monde réel et résolvent les problèmes de dessin. Elle comporte trois éléments principaux : un environnement de développement, des modèles de dessins et une suite d'outils divers.

Ces outils permettent la manipulation du dessin et de l'interface utilisateur. Certains sont similaires à ceux proposés dans les autres programmes Microsoft Office tandis que d'autres sont spécifiques à Visio : les touches de raccourci, la personnalisation des menus, la correction automatique, le vérificateur orthographique, le système d'aide, l'enregistrement en tant que page Web, la recherche de formes, le choix dans la disposition et le type de fenêtres d'affichage, l'insertion de commentaires et d'images, la définition de styles, la collaboration en ligne, l'envoi de dessins par courrier électronique, la génération automatique de rapports, l'importation et l'exportation de données entre le dessin et des sources externes, etc.

L'environnement de développement Visio fournit une série d'outils de programmation pour créer des solutions personnalisées :

- VBA (Microsoft Visual Basic for Applications) qui permet de créer des macros et de contrôler Visio à l'aide d'Automation;
- la bibliothèque de types Visio qui décrit les objets, propriétés, méthodes, événements et constantes que Visio expose aux clients d'Automation;
- la bibliothèque d'objets Visio qui fournit aux clients d'Automation des informations sur les objets, propriétés, méthodes, événements et constantes disponibles;
- la feuille ShapeSheet qui permet de créer des formules pour déterminer l'aspect et le comportement de formes spécifiques.

Ces outils manipulent les objets Visio et permettent donc d'écrire des programmes qui intègrent la fonctionnalité de Visio. Ils peuvent servir à automatiser certaines tâches et à modifier l'interface utilisateur. Ces programmes, appelés macros, doivent être écrits dans un langage de programmation qui prend en charge Automation, ce qui est le cas de VBA dont le code est rédigé au moyen de l'éditeur Visual Basic associé à Visio.

La hiérarchie d'objets ou le modèle objet du programme fait référence à la façon dont les objets sont reliés entre eux, ainsi qu'aux propriétés (données), méthodes (comportements) et événements de chaque objet. La bibliothèque d'objets Visio fournit aux contrôleurs d'Automation (comme VBA) des informations sur les objets disponibles, tandis que la bibliothèque de types contient la description standard des objets, des propriétés et des méthodes disponibles pour Automation. Le projet VBA d'un document Visio fait automatiquement référence à ces bibliothèques. Il comprend l'ensemble des macros[9] du document et se présente sous la forme d'un jeu de modules. Un module est constitué de déclarations suivies de procédures et/ou de fonctions. Il peut être de deux types : un module de classe qui contient la définition d'une classe, notamment les définitions de ses propriétés et de ses méthodes, et un module standard qui contient uniquement des définitions et des déclarations de procédures, de fonctions, de types et de données.

Si l'Automation permet à un programme de contrôler les dessins Visio en accédant à ses objets, puis en manipulant ses propriétés, ses méthodes et ses événements, il permet également d'utiliser des objets provenant d'autres applications en code VBA. Celles-ci communiquent des informations sur leurs objets grâce à leurs propres bibliothèques de types et d'objets.

Voici une brève présentation des éléments intervenant dans l'Automation :

- les objets représentent des éléments manipulables tels que des documents, des pages de dessin, des formes et des cellules qui contiennent des formules. La plupart d'entre eux correspondent à des éléments visibles et sélectionnables dans l'interface utilisateur;
- les propriétés sont des attributs qui déterminent l'apparence ou le comportement des objets. Par exemple, un objet "Shape" (forme) possède une propriété "Name" qui représente le nom de cette forme;
- les méthodes sont des actions associées à un objet. Par exemple, une forme peut être crée, modifiée, supprimée ou elle peut renvoyer son type, des données ou le bloc de texte qui lui sont associés;
- les événements sont des occurrences ou des notifications qui peuvent déclencher des réponses. Par exemple, l'ouverture d'un document ou le double-clic sur une forme peut déclencher l'exécution d'un code (macro) ou de programmes entiers.

Troisième et dernier élément de la solution Visio, le modèle de dessin inclut tout ce dont l'utilisateur a besoin pour créer un type de dessin particulier : une page de dessin vierge possédant une grille et un système de mesure appropriés au type de dessin qu'il souhaite créer, un ou plusieurs gabarits contenant des formes, des éléments de programmation (macros) et, enfin, divers paramètres tels qu'une échelle et une taille de page correctes. L'utilisation d'un modèle, défini par Visio ou construit par l'utilisateur, garantit une certaine cohérence parmi les projets standards. Ce modèle constitue alors l'espace de travail de l'utilisateur.

Un gabarit est une collection de formes de base associée à un modèle Visio. Lorsque l'on glisse une forme de base à partir d'un gabarit sur la page de dessin, la forme créée devient une instance de la forme de base. Cette instance hérite de la mise en forme et des autres propriétés de sa forme de base, tandis que celle-ci reste sur le gabarit et peut donc être réutilisée pour créer de nouvelles instances.

[9] Une macro est une procédure ou une fonction pouvant contenir des instructions et d'autres procédures ou fonctions. Elle permet d'exécuter automatiquement des tâches répétitives.

En pratique, le terme "forme" se rapporte à tout ce qui se trouve sur la page de dessin :

- les formes insérées à partir des gabarits comme expliqué ci-dessus;
- les traits des liens reliant les formes;
- les blocs de texte : un bloc de texte est une zone de texte qui est associée à une forme et qui est habituellement visible au centre de cette forme;
- les formes tracées à main levée à l'aide des outils de dessin;
- les objets importés, incorporés ou liés provenant d'autres programmes.[10]

La création d'une forme s'effectue soit en glissant une forme de base d'un gabarit, soit en la dessinant à main levée, soit en la dupliquant (copier/coller une forme déjà présente sur le dessin). La partie projet du graphe de la figure 2.1 ne fait référence qu'à la première technique.

L'apparence d'une forme est déterminée par son formatage. Ce dernier est défini par de nombreux paramètres individuels que l'on peut diviser en trois catégories : le remplissage, le trait et le texte. Tous les types de formatage ne peuvent pas être appliqués à toutes les formes. Par exemple, ni les traits, ni les blocs de texte, ni les objets incorporés ne peuvent recevoir un format de remplissage.

On distingue les formes unidimensionnelles (1D) et les formes bidimensionnelles (2D). Les premières se comportent comme des traits : elles ne possèdent que deux extrémités. Les secondes peuvent être simples comme un rectangle ou complexes comme le dessin d'un bureau. Elles peuvent également être composées de formes regroupées.

Chaque forme possède un certain nombre (deux pour les formes unidimensionnelles) de poignées de sélection que l'on peut faire glisser pour la redimensionner ou la repositionner sur le dessin. L'extrémité d'une forme 1D peut être collée sur l'un des points de connexion situés au bord d'une forme 2D. L'utilisateur peut ainsi créer un trait de liaison entre deux formes bidimensionnelles.

Un dessin Visio peut constituer un support de stockage de données : une forme contient des propriétés personnalisées qui lui permettent de stocker des données en saisissant des valeurs dans une boîte de dialogue ou dans une fenêtre. L'utilisateur peut modifier les propriétés personnalisées des formes existantes, en ajouter de nouvelles ou en supprimer, de même qu'il peut en insérer dans les formes qu'il crée. Les propriétés personnalisées possèdent leur propre section dans la feuille de calcul ShapeSheet (voir ci-dessous) où l'utilisateur a la possibilité de les manipuler directement.

Le gabarit de document – différent du gabarit évoqué précédemment – contient l'inventaire des formes de base utilisées dans tous les dessins du document : chaque fois qu'une forme de base est glissée sur la page de dessin à partir d'un gabarit, elle est copiée dans le gabarit de document et une instance de celle-ci est créée sur la page de dessin. Cette instance hérite de la mise en forme et des autres propriétés de la forme de base correspondante dans le gabarit et, par conséquent, dans le gabarit de document. Mais ce dernier conserve une trace des formes de base originales utilisées dans le dessin, même lorsque les formes de base du gabarit ont été modifiées ou supprimées. Ce lien d'héritage permet de modifier toutes les instances d'une forme de base d'un document en révisant celle-ci dans le gabarit de document. Ce procédé est transparent pour l'utilisateur. Le gabarit de document, situé dans la partie projet du graphe de la figure 2.1, n'y apparaît pas pour cette raison et pour ne pas complexifier davantage le schéma. En pratique, si la forme hérite des caractéristiques de la forme de base

[10] Dans notre mémoire, nous ne rencontrerons pas ces deux derniers types de forme.

du gabarit, les liaisons d'instanciation doivent associer la forme et la plupart de ses composants (propriétés personnalisées, formatage et feuille ShapeSheet) à la forme de base du gabarit de document et à ses composants correspondants. Le calque, qui est un concept indépendant des formes, n'est pas concerné par la modification ou la disparition des formes de base : le calque du projet, modifiable, reste une instance du calque de l'espace de travail.

Les calques sont des catégories de formes pourvues d'un nom. Ils permettent d'organiser des formes apparentées sur une page de dessin : en associant les formes à différents calques, il est possible d'afficher, d'imprimer, de colorer et de verrouiller séparément différentes catégories de formes, mais également de définir ou non l'alignement ou le collage des formes d'un calque. Une forme (de base) peut être attribuée à un, à plusieurs ou à aucun calque. Dans le graphe de la figure 2.1, les calques du projet sont hérités des calques de l'espace de travail. Mais ils peuvent aussi être modifiés et de nouveaux calques spécifiques au projet peuvent également être créés.

Comme tout objet de Visio (groupe, style, page ou document), chaque forme possède une feuille de calcul ShapeSheet dans laquelle sont stockées les informations qui la concernent (angle, dimensions, centre de rotation, protection, bloc de texte, informations sur le collage, événements gérés, type de connexions entre les formes, propriétés personnalisées, etc.), ainsi que les styles qui déterminent son aspect (trait, remplissage, format du bloc de texte, etc.).

La feuille ShapeSheet est divisée en sections qui commandent chacune un aspect particulier du comportement ou de l'aspect d'une forme. Une section contient une ou plusieurs lignes qui incluent à leur tour des cellules. Chaque cellule peut contenir une formule (obligatoire ou facultative), son résultat (appelé "valeur" de la cellule), et, le cas échéant, des informations sur les erreurs éventuelles.[11] Les données d'une cellule (sa formule ou sa valeur) peuvent être définies localement ou, plus généralement, héritées de la formule équivalente de la forme de base correspondante. Concrètement, la fenêtre de dessin Visio et la fenêtre de la feuille ShapeSheet constituent simplement des vues différentes de la même forme de sorte que lorsque celle-ci est modifiée, Visio met automatiquement à jour les formules contenues dans la feuille.

La feuille de calcul ShapeSheet est aussi considérée comme un outil de développement puisqu'elle contient des formules que l'utilisateur peut définir – soit en accédant directement à la fenêtre de la feuille ShapeSheet, soit par programmation – et qui représentent les attributs déterminant l'aspect et le comportement de la forme. Exemples : la taille de la forme peut s'adapter à la largeur de son bloc de texte, un événement comme la modification du texte de la forme peut provoquer l'exécution d'une macro, les formules de certaines cellules peuvent être modifiées en fonction des données introduites par l'utilisateur dans les propriétés personnalisées.

L'éditeur graphique Visio présente un grand intérêt pour nous parce qu'il nous offre la possibilité de construire notre propre modèle réutilisable en choisissant des gabarits, des formes et des macros déjà présents dans la solution Visio, mais aussi en nous permettant de définir nos propres gabarits, nos propres formes avec des formatages particuliers, ainsi que nos propres macros ou programmes au moyen de VBA ou d'applications étrangères compatibles.

[11] Visio considère tout ce qui apparaît dans une cellule – même une valeur numérique ou une simple référence à une autre cellule – comme une formule.

Chapitre 3 :

La modélisation graphique des "parts"

Ce chapitre touche au cœur de l'objet de notre mémoire : la création de modèles graphiques de "parts" et leur traduction en langage AlbertII. Jusqu'à présent, nous n'avions jamais quitté un certain niveau de conceptualisation avec la présentation de la thèse et du logiciel Microsoft Visio. Nous allons maintenant mettre en pratique tous ces préceptes. Nous combinerons les aspects théoriques relatifs au pattern *Part* et au langage AlbertII et les aspects fonctionnels liés à la modélisation et à l'édition graphique.

La première partie est consacrée à la modélisation du pattern *Part* qui a été présenté dans le premier chapitre. Nous nous attacherons d'abord à déterminer un sous-ensemble de ce pattern en redéfinissant ses classes générales, ses méta-classes et ses classes spéciales. Nous avons en effet souhaité le simplifier et l'adapter à la modélisation graphique et aux possibilités du logiciel Visio. Nous établirons d'ailleurs un parallèle entre ce sous-ensemble et les concepts présentés dans le deuxième chapitre. Nous y inclurons une description du modèle du système manufacturier et de son fonctionnement dans Visio. Nous insisterons notamment sur les règles de réutilisation du sous-ensemble. Nous passerons ensuite en revue les différentes contraintes qui découlent du pattern *Part* mais qui résultent aussi de son adaptation au logiciel Visio. Nous verrons comment la majorité d'entre elles peuvent être vérifiées par la modélisation graphique ou traduites en AlbertII. Cet exposé sera accompagné de quelques remarques critiques et constructives à l'égard du pattern *Part* défini dans la thèse.

La deuxième partie du chapitre identifie une fraction du langage AlbertII qui sera adaptée au sous-ensemble défini précédemment. Nous décrirons d'abord ses caractéristiques principales. Nous donnerons ensuite sa syntaxe concrète que nous avons voulu aussi proche que possible du code généré par l'éditeur Visio.

La troisième partie présente les règles de mappage qui président à la traduction des modèles graphiques de "parts" en langage AlbertII. Nous expliquerons comment faire la correspondance entre, d'une part, les méta-classes et les classes spéciales du sous-ensemble du pattern *Part* et, d'autre part, les types de données AlbertII. Nous exposerons également la méthode employée pour exprimer les classes spécifiques en types de données spécifiques de base et construits. Nous enseignerons aussi la manière de traduire certaines contraintes en langage AlbertII.

Nous terminerons par l'annexe F et la présentation du code du projet VBA attaché au modèle Visio du système manufacturier et hérité par tous les projets de dessin particuliers. Nous exposerons quelques concepts importants relatifs au langage Visual Basic for Applications employé dans Visio puis nous expliciterons l'algorithme général chargé de personnaliser l'espace de travail, de vérifier les contraintes attachées aux modèles graphiques de "parts" et de traduire ces modèles en langage AlbertII.

3.1. La modélisation du pattern *Part*

3.1.1. Définition d'un sous-ensemble du pattern *Part*

La démarche qui sous-tend la réalisation de notre mémoire n'exige pas la reprise du pattern *Part* dans son intégralité : la construction de modèles graphiques de "parts" est réalisable avec un nombre limité de classes de ce pattern. Nous n'emploierons pas les classes de parts existantes, ainsi que les classes d'attributs dispositifs de fixation qui sont associées aux classes de parts composées et qui représentent des propriétés optionnelles sur les parts. Nous délaisserons également les classes spéciales de tampons vides, de conteneurs vides et de contenants, ainsi que la moitié des classes spéciales d'attributs. Les contraintes relatives à ces classes seront aussi abandonnées.

D'autre part, certaines classes reprises seront modifiées ou mises au second plan pour être adaptées à la modélisation graphique. Nous ferons de même avec les contraintes y afférentes. Toutefois, nous conserverons les caractéristiques obligatoires (attributs et contraintes) qui s'appliquent à toutes les parts d'une application, indépendamment de la classe spécifique à laquelle elles peuvent appartenir, et qui doivent nécessairement être présentes dans chaque modèle.

Nous avons choisi de conserver – dans la mesure du possible – les noms anglais des classes d'origine du pattern *Part* afin de marquer la filiation de notre sous-ensemble avec ce pattern défini dans la thèse. [Petit 1999, pp. 183-204] Nous continuerons aussi à transcrire le nom des classes en italique.

Pour la description détaillée des caractéristiques des parts en général et des classes du sous-ensemble en particulier, nous renvoyons le lecteur à la troisième partie du premier chapitre consacrée au pattern *Part*.

3.1.1.1 Les classes générales

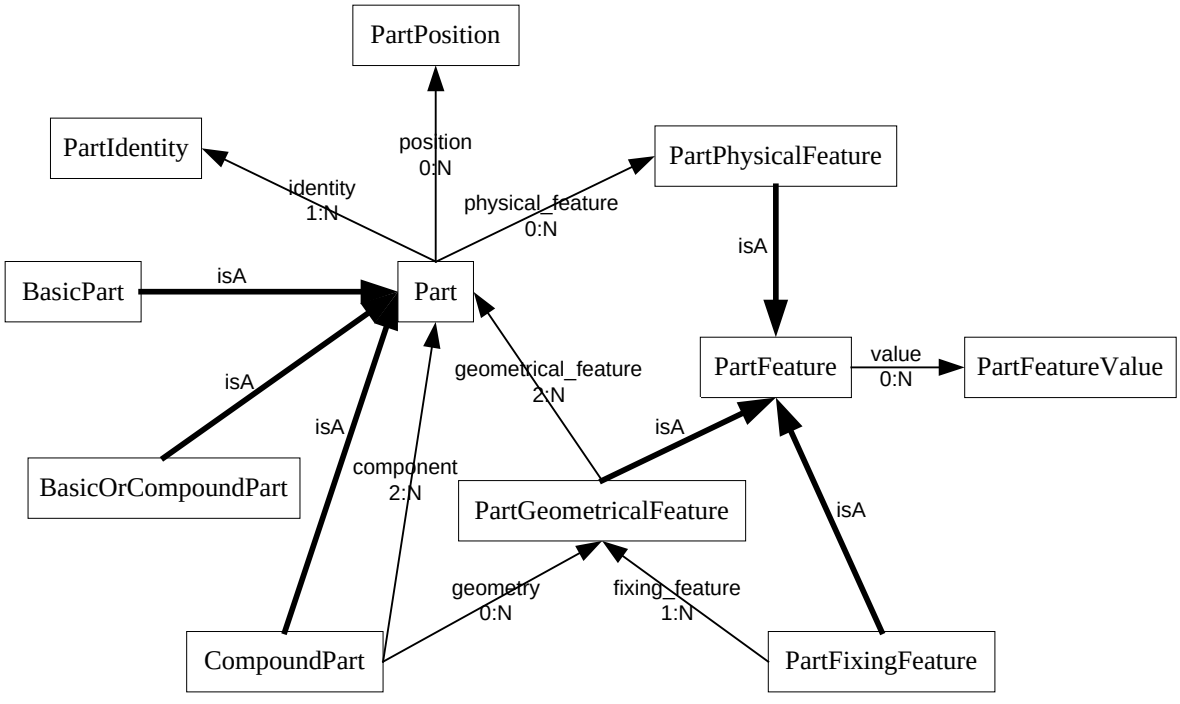


Figure 3.1 : Description graphique des classes générales du sous-ensemble du pattern *Part*

La figure 3.1 présente le graphe des classes générales du sous-ensemble du pattern *Part*. Conformément à ce qui a été évoqué ci-dessus, il ne comprend pas la classe générale des parts existantes, ni les classes générales d'attributs dispositifs de fixation.

Cinq différences sont visibles par rapport au graphe des classes générales du pattern *Part* exposé à la figure 1.2. La première est l'absence de la classe générale des parts existantes (*ExistingPart*), sous-ensemble de la classe générale de parts (*Part*). Dans la thèse, cette classe traduit le principe selon lequel l'ensemble des parts existant à un moment donné constitue un sous-ensemble de l'ensemble des parts concevables. [Petit 1999, p. 185] Cela induit une hiérarchisation des classes générales à trois niveaux comme indiqué de manière simplifiée à la figure 3.2.

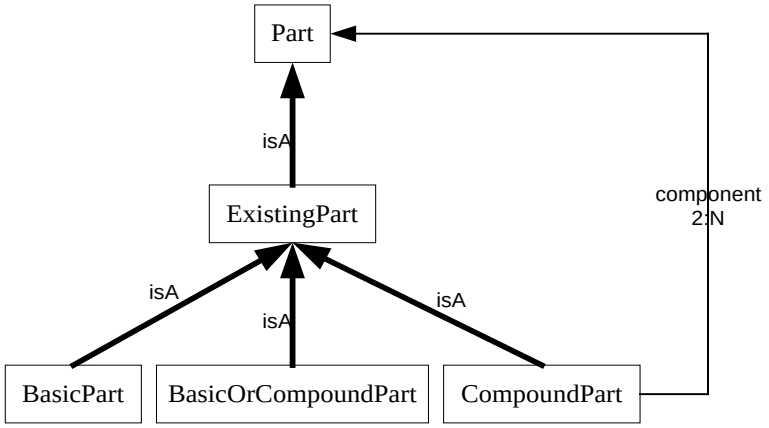


Figure 3.2 : Description graphique simplifiée de la hiérarchie des classes générales du pattern *Part*

Or, le fait de considérer les parts comme existantes ou non relève selon nous d'une simple vue de l'esprit. Nous pouvons en effet raisonnablement penser qu'un système manufacturier n'emploie que des parts existantes. Cela revient à supprimer le niveau intermédiaire composé de la seule classe générale des parts existantes. Par conséquent, toutes les parts instanciées des classes spécifiques appartenant aux modèles graphiques de "parts" sont considérées implicitement comme des parts existantes.

La disparition de la classe générale des parts existantes a une incidence sur la classe générale d'identités (*PartIdentity*) en vertu de la (double) contrainte (n° 7) qui veut que, sur l'ensemble des parts existantes, deux parts ne puissent posséder la même identité. Cette contrainte devient simplement : toutes les parts – et non plus toutes les parts existantes – possèdent des identités distinctes. La contrainte d'unicité reste mais celle qui exigeait qu'une part doive exister pour posséder une identité disparaît. Cette simplification est d'ailleurs le but recherché par la suppression de cette classe générale. En conséquence, chaque part doit posséder au moins une identité (et cette identité doit être unique).[12] L'attribut identité qui était

[12] Nous imposons aussi une restriction à l'identité d'une part. Dans la thèse, il est stipulé que chaque part doit posséder une identité distincte mais que celle-ci peut être dérivée de tout ou partie de son état. Une identité n'est donc pas nécessairement déterminée par les instances des classes d'identités, sous-entendu qu'une classe de parts peut ne pas être associée à une classe d'identités. Or, nous imposons que cette identité soit dorénavant uniquement définie par un ensemble d'identifiants appartenant chacun à une instance différente de la méta-classe d'identités.

un état facultatif pour une part devient donc obligatoire : la cardinalité de la classe générale d'attributs identités (*identity*) est modifiée pour être fixée à "1:N". Ceci constitue la seconde différence par rapport au pattern *Part*.

Cependant, le caractère obligatoire de l'attribut identité induit qu'une part doit être associée à une et une seule instance de chacune des classes spécifiques d'identités liées à la classe spécifique de parts dont elle est l'instance. La classe générale d'identités ne se comportait pas autrement dans le pattern *Part* : chaque instance d'une classe de parts devait être associée à une instance de chaque classe d'identités liée à cette classe de parts et la contrainte en langage AlbertII exprimait que les instances des classes d'attributs identités qui effectuaient la liaison (entre la classe de parts et les classes d'identités) ne pouvaient pas renvoyer une valeur d'identité indéfinie. Cette contrainte disparaît maintenant car elle est automatiquement vérifiée par la cardinalité imposée ("1:1") aux classes spécifiques d'attributs identités.

La troisième différence avec le pattern *Part* réside dans l'apparition d'une nouvelle classe : la classe générale de parts de base ou composées, appelée aussi classe générale de parts mixtes (*BasicOrCompoundPart*). Cette classe a été conçue pour répondre à une critique du modèle de *Part* tel qu'il est présenté dans la thèse. Voir la partie de la section suivante consacrée à la méta-classe de parts mixtes (*BasicOrCompoundPartClass*).

Pour rendre la construction des modèles graphiques de "parts" moins complexe, nous avons sacrifié une des quatre classes générales d'attributs spécifiques aux classes générales de dispositifs géométriques (*PartGeometricalFeature*) et de fixation (*PartFixingFeature*). Nous avons conservé les deux classes générales d'attributs concernés (*concerns*) qui définissent les classes de dispositifs géométriques et de fixation parce qu'elles nous paraissaient plus significatives que les classes générales d'attributs dispositifs géométriques (*geometrical_feature*) et de fixation (*fixing_feature*) qui définissent les classes de parts composées. Les dispositifs géométriques indiquent en effet la manière dont les composants d'une part sont assemblés : ils se rapportent donc davantage aux parts composantes qu'aux parts composées. Quant aux dispositifs de fixation, ils précisent le moyen employé pour assembler les composants et intéressent plutôt les dispositifs géométriques que les parts composées.

Dans un premier temps, nous avions également supprimé la classe générale d'attributs dispositifs géométriques mais certaines configurations ne permettaient pas de savoir à quelles classes de parts composées étaient attachées certaines classes de dispositifs géométriques. C'était par exemple le cas lorsqu'une classe de parts composantes était associée à deux classes de parts composées et à deux classes de dispositifs géométriques qui ne possédaient chacune qu'une seule classe d'attributs dispositifs géométriques : le modèle ne permettait pas de déterminer à quelle classe de parts composées se rapportait chacune des classes de dispositifs géométriques. Les classes de dispositifs de fixation ne sont pas concernées par ce problème car, en vertu des troisième et quatrième contraintes sur les méta-classes (du sous-ensemble) du pattern *Part*, elles doivent être associées à des classes de dispositifs géométriques qui ne définissent qu'une et une seule classe de parts composées. Ainsi, le sous-ensemble du pattern *Part* conserve implicitement la liaison matérialisée par la classe supprimée. Les classes générales d'attributs dispositifs géométriques, concernés et dispositifs de fixation forment en effet les côtés d'un triangle dont les sommets sont les classes générales de parts composées, de dispositifs géométriques et de dispositifs de fixation. Même si nous supprimons un côté, nous conservons toujours la liaison entre les trois sommets. Cependant, nous perdons les informations données par la cardinalités de la classe d'attributs dispositifs de fixation, à savoir les nombres minimum et maximum de dispositifs de fixation associés à une part composée.

Afin de différencier les deux classes générales d'attributs concernés au sein du sous-ensemble du pattern *Part*, nous leur avons attribué les noms des deux classes antérieurement effacées : la nouvelle classe générale d'attributs dispositifs géométriques (*geometrical_feature*) relie la classe générale de dispositifs géométriques à la classe générale de parts, tandis que la nouvelle classe générale d'attributs dispositifs de fixation (*fixing_feature*) unit la classe générale de dispositifs de fixation à la classe générale de dispositifs géométriques. Ces noms sont plus explicites et rappellent celui de la classe générale de dispositifs physiques (*physical_feature*). Quant à la classe "ressuscitée", elle porte désormais le nom de "classe générale d'attributs géométries" (*geometry*).

Nous aurions voulu inverser le sens de la nouvelle relation dispositif géométrique pour effectuer un parallèle avec la relation dispositif physique qui aboutit au dispositif physique et pour faire du dispositif géométrique une propriété de la part car celle-ci est l'élément central du modèle. D'autre part, cette inversion aurait permis une traduction uniforme des dispositifs de parts simple et évalué en langage AlbertII puisque le dispositif géométrique simple, comme le dispositif physique simple, aurait été traduit en un type de base tandis que le dispositif géométrique évalué, à l'instar du dispositif physique évalué, aurait été traduit en un type construit sous la forme d'un produit cartésien. Cependant, la contrainte de géométrie présentée dans les sections suivantes n'aurait pas pu être exprimée en AlbertII car elle aurait dû prendre en considération des classes d'attributs dispositifs géométriques appartenant à des types construits différents (ceux qui correspondent aux différentes classes de parts composantes).

La suppression de la classe générale d'attributs dispositifs de fixation, ainsi que le changement de nom de la classe générale d'attributs dispositifs géométriques et des deux classes générales d'attributs concernés, constituent les deux dernières différences par rapport au graphe des classes générales du pattern *Part* décrit dans la thèse.

Les contraintes de cardinalité sur les attributs sont exprimées sur le graphe de la figure 3.1 sous la forme de cardinalités minimales et maximales. Les contraintes sur les classes générales sont les mêmes que celles présentées au premier chapitre. Les contraintes n° 4 à 6 ont été légèrement modifiées pour être adaptées au sous-ensemble :

1. La classe générale des parts de base (*BasicPart*) et la classe générale des parts composées (*CompoundPart*) forment une partition de la classe générale des parts : une part ne peut pas être à la fois une part de base et une part composée mais elle doit être l'une des deux.
2. Aucune part n'est composée d'elle-même.
3. Les classes générales de dispositifs physiques (*PartPhysicalFeature*), géométriques (*PartGeometricalFeature*) et de fixation (*PartFixingFeature*) forment une partition de la classe générale des dispositifs : un dispositif ne peut appartenir qu'à une et une seule de ces trois classes.
4. Chaque dispositif géométrique concerne (assemble) uniquement les composants directs d'une même part composée.
5. Chaque dispositif de fixation concerne (fixe) uniquement les dispositifs géométriques qui assemblent les composants directs d'une même part composée.
6. Toutes les parts possèdent des identités distinctes (appartenant aux instances de la méta-classe d'identités).
7. Aucune part ne peut être le composant de deux parts composées.

La manière de représenter les dispositifs de part simples (*simple part feature*) et les dispositifs de part évalués (*valued part feature*) au sein du pattern *Part* (voir la figure 2.1) et de son sous-ensemble (voir la figure 3.1) est critiquable. En effet, les seconds se distinguent des premiers par l'obligation de posséder une ou plusieurs valeurs s'ils sont eux-mêmes associés à une part.[13] Or, la cardinalité ("0:N") de la classe générale d'attributs valeurs (*value*) permet de définir une classe spécifique d'attributs valeurs dont la borne inférieure de la cardinalité est nulle.[14] Dans ce cas, nous pouvons obtenir des dispositifs de part évalués sans valeurs car celles-ci sont facultatives. Au niveau du pattern *Part*, une contrainte formalisée en langage AlbertII permet d'éviter ce genre d'erreur. Dans notre sous-ensemble, une macro vérifie que la borne en question n'est jamais nulle.

La solution pour rendre les deux graphes moins ambigus serait de créer deux classes générales pour chaque type de dispositifs – une pour les dispositifs de part simples et une pour les dispositifs de part évalués – et d'associer directement la seconde à la classe générale de dispositifs de valeur par l'intermédiaire d'une classe générale d'attributs valeurs dont la borne inférieure de la cardinalité ne serait pas nulle ("1:N"). C'est la philosophie prônée par la modélisation graphique des "parts". Elle est développée dans la section consacrée à la correspondance entre le sous-ensemble du pattern *Part* et les concepts Visio.

3.1.1.2 Les méta-classes

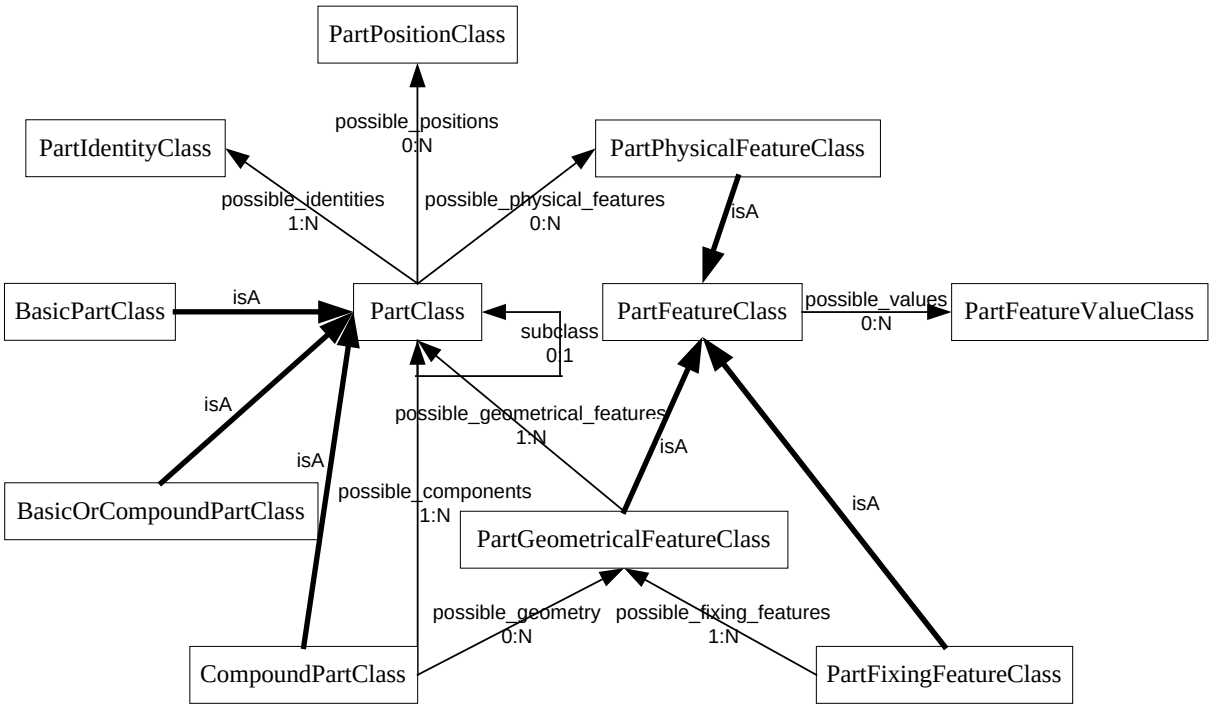


Figure 3.3 : Description graphique des méta-classes du sous-ensemble du pattern *Part*

[13] Au niveau des modèles de "parts", cette différence se traduit par le fait qu'une classe de dispositifs de valeur n'est jamais associée à une classe de dispositifs de part simples tandis qu'une classe de dispositifs de part évalués est toujours unie à au moins une classe de dispositifs de valeur par l'intermédiaire d'une classe d'attributs valeurs dont la borne inférieure de la cardinalité ne peut être nulle.

[14] La borne inférieure de la cardinalité de la classe générale d'attributs valeurs doit être nulle car les classes de dispositifs de part simples ne peuvent pas être associées à une classe de dispositifs de valeur.

La figure 3.3 présente le graphe des méta-classes du sous-ensemble du pattern *Part*. La méta-classe des parts existantes, ainsi que la méta-classe d'attributs dispositifs de fixation, ont été retirées.

En raison de la correspondance avec les classes générales, les mêmes modifications ont été opérées sur le schéma : la méta-classe des parts existantes a disparu puisque toutes les classes spécifiques de parts créées dans les modèles graphiques de "parts" appartiennent désormais implicitement à la classe générale des parts existantes. Comme expliqué à la section précédente, chaque part doit posséder une identité (représentée par un ensemble d'identifiants), d'où l'obligation d'associer au moins une classe d'identités à toutes les classes de parts. La cardinalité de la méta-classe d'attributs identités a donc été modifiée pour prendre la valeur "1:N".

La méta-classe d'attributs dispositifs de fixation du pattern *Part* a été supprimée, tandis que la méta-classe d'attributs dispositifs géométriques et les deux méta-classes d'attributs concernés ont reçu chacune une nouvelle dénomination à la manière des classes générales d'attributs correspondantes. Dorénavant, dans le sous-ensemble, la méta-classe d'attributs géométries (*possible_geometry*) associe la méta-classe de parts composées (*CompoundPartClass*) à la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*), tandis que la méta-classe d'attributs dispositifs géométriques (*possible_geometrical_features*) relie la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*) à la méta-classe de parts (*PartClass*), alors que la méta-classe d'attributs dispositifs de fixation (*possible_fixing_features*) unit la méta-classe de dispositifs de fixation (*PartFixingFeatureClass*) à la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*).

Pour éviter de complexifier inutilement la construction et l'analyse des modèles graphiques de "parts", la sémantique des classes dans une hiérarchie de spécialisation a été définie comme une partition : une super classe abstraite constituée d'un ensemble de sous-classes exclusives. Ce principe ne constitue en rien une restriction dans la définition des modèles de "parts" car une hiérarchie composée de sous-classes exclusives sans super classe abstraite ou d'une super classe abstraite sans sous-classes exclusives peut facilement être transformée en une hiérarchie dotée d'une liaison de spécialisation unique qui est une partition : il suffit de grouper et/ou de subdiviser les instances des classes de parts jusqu'à ce que chaque instance de la super classe n'appartienne plus qu'à une et une seule des sous-classes de cette super classe. Cette technique doit être appliquée entre deux niveaux consécutifs d'une même hiérarchie.

Le modèle du pattern *Part* dans la thèse ne permet pas à une classe de parts de base d'être le sous-ensemble d'une classe de parts composées. En effet, comme les sous-classes héritent des classes d'attributs de leurs super classes, la classe de parts de base devrait hériter des classes d'attributs composants de sa super classe de parts composées. Or, cela contrevient à la définition de la part de base qui stipule qu'une part de base ne peut être constituée d'autres parts. L'inverse est également vrai : une classe de parts composées ne peut pas être le sous-ensemble d'une classe de parts de base. En effet, si une instance de la super classe de parts de base est du type de sa sous-classe de parts composées, la contrainte n° 1 relative aux classes générales – une part ne peut pas être à la fois une part de base et une part composée – n'est pas respectée. Cette remarque est également valable si une classe de parts composées est la super classe d'une classe de parts de base.

Pour résoudre cette contrainte qui provoque indûment l'uniformisation des hiérarchies de sous-ensembles, plusieurs solutions s'offrent à nous. La première consiste à interdire à une super classe de parts composées de posséder des sous-classes de parts de base, sous peine de contrevenir au principe selon lequel une classe de parts de base ne peut hériter de classes d'attributs composants, mais à autoriser une super classe de parts de base à disposer de sous-classes de parts composées. Dans ce cas, on tolère une exception à la contrainte n° 1 relative aux classes générales, ce qui n'est guère gênant. Cependant, l'usage d'exceptions n'est jamais conseillé et une dissymétrie apparaît dès lors entre les classes de parts parce que le phénomène d'uniformisation persiste lorsque la super classe est une classe de parts composées.

La seconde solution consiste à faire disparaître la distinction entre les classes de parts. Les méta-classes et les classes générales de parts de base et de parts composées sont donc supprimées, de même que la contrainte n° 1 relative aux classes générales. Les nouvelles parts doivent se comporter à la fois comme les anciennes parts de base et comme les anciennes parts composées, notamment au niveau des relations de composition : à tout moment, elles doivent être constituées soit d'aucune, soit d'au moins deux autres parts. La contrainte d'uniformité est totalement levée mais cette solution introduit un profond bouleversement de nature à modifier l'essence même du pattern *Part*. D'autre part, cela complique la lecture et la construction des modèles graphiques de "parts" parce qu'on ne sait plus distinguer si une part possède ou non des composants. Or, cette distinction est d'autant plus importante qu'elle permet de placer correctement les dispositifs géométriques qui sont exclusivement employés dans les relations de composition. En outre, cela introduit la possibilité d'attribuer puis de retirer des composants à une part et ce changement de statut (d'une part) doit être géré par des contraintes en langage AlbertII, d'où une complexité croissante.[15]

La troisième solution, que nous avons adoptée, préconise la création d'une méta-classe de parts de base ou composées (*BasicOrCompoundPartClass*), appelée aussi méta-classe de parts mixtes, ainsi que d'une classe générale correspondante (*BasicOrCompoundPart*). Celle-ci est définie comme une instance de la méta-classe de parts mixtes, tandis que toutes les instances de cette méta-classe sont interprétées comme une spécialisation de la classe générale de parts mixtes dont elles héritent des propriétés. L'usage des classes spécifiques de parts mixtes dans les modèles graphiques de "parts" est très restrictif : elles doivent uniquement être employées dans le cas de relations de composition et ce, dans des configurations bien particulières. Si nous représentons la hiérarchie des sous-ensembles comme un arbre, une classe de parts mixtes ne peut être qu'une super classe qui possède au moins, soit une sous-classe directe de parts mixtes, soit une sous-classe directe de parts de base et une sous-classe directe de parts composées. D'autre part, une classe de parts mixtes ne peut être la sous-classe que d'une autre classe de parts mixtes. L'objectif poursuivi est de regrouper des classes de parts de base et des classes de parts composées au sein d'un même arbre sans qu'il subsiste une filiation directe entre ces deux types de classes de parts.

Nous levons donc partiellement la contrainte en permettant l'unification de sous-arbres uniformes au moyen de classes de parts mixtes. Sa suppression complète est irréalisable car, comme nous l'avons expliqué ci-dessus, elle entrerait en contradiction avec la définition des types de parts : une part composée est un agrégat formé d'un ensemble d'autres parts tandis qu'une part de base est une part atomique.

[15] La présence et l'absence de relations de composition à des intervalles de temps différents exercent notamment une influence sur le comportement des dispositifs géométriques et de fixation.

L'utilisation limitée des classes de parts mixtes fait que, dans une hiérarchisation de spécialisation hétérogène, seules les classes de parts de base et les classes de parts composées constituent les feuilles de l'arbre, la racine étant toujours une classe de parts mixtes. Puisque nous avons indiqué par ailleurs que la sémantique des classes dans une telle hiérarchie est toujours définie comme une partition, chaque instance d'une classe spécifique de parts mixtes est toujours une part mixte mais aussi une part de base ou une part composée sans être les deux à la fois. À l'échelle de n'importe quel modèle graphique de "parts", une part est toujours soit une part de base, soit une part composée, soit à la fois une part de base et une part mixte, soit à la fois une part composée et une part mixte. Nous pouvons donc doublement conclure que, d'une part, une part ne peut être à la fois une part de base et une part composée et que, d'autre part, une part doit être au moins une part de base ou une part composée. Nous entendons ainsi respecter l'esprit du principe de partition de la classe des parts (contrainte n° 1 sur les classes générales) : une part ne peut pas être à la fois une part de base et une part composée mais elle doit être l'une des deux. Nous considérons en quelque sorte qu'une classe de parts mixtes, qui est une classe abstraite, n'existe qu'au travers de ses sous-classes feuilles : le type de l'instance d'une super classe de parts mixtes est déterminé par le type d'une de ses sous-classes de parts feuilles, celle-ci étant toujours soit une classe de parts de base, soit une classe de parts composées. D'ailleurs, lorsque les classes de parts sont traduites en types de données AlbertII, ceux-ci reconnaissent comme types les parts de base et les parts composées, et non les parts mixtes.

Le respect de la définition de la part de base impose que cette dernière ne puisse pas hériter des relations composants au sein d'une hiérarchie de sous-ensembles. Les classes de parts mixtes ne peuvent donc pas posséder de classes d'attributs composants.

Voici un exemple illustrant la simplicité de la troisième solution par rapport à la deuxième. Supposons une hiérarchie de spécialisation composée de trois classes : une super classe de parts composées (*bloc_moteur*) associée à une classe de parts composantes (*piston*), une sous-classe de parts de base (*bloc*) et une sous-classe de parts composées (*bloc_pistons*). Voyons ce qui se passe lorsque nous appliquons la deuxième solution qui ne fait plus la distinction entre les classes de parts de base et les classes de parts composées. Puisque *bloc* est une classe de parts de base, par définition, elle ne peut pas posséder de composants pistons. Un algorithme doit donc vérifier au préalable que la borne inférieure de la cardinalité de la classe d'attributs composants est nulle. Si *bloc_moteur* est de type *bloc*, une contrainte doit indiquer que la cardinalité de cette classe est obligatoirement nulle. Par contre, si le type de *bloc_moteur* est *bloc_pistons*, une autre contrainte doit signaler que la même cardinalité doit être égale ou supérieure à deux. La troisième solution permet d'envisager ce cas de figure différemment : la super classe devient une classe mixte (elle ne possède donc plus de classe d'attributs composants), la sous-classe *bloc_pistons* est maintenant associée à la classe de parts composantes *piston* et la sous-classe *bloc* n'est pas modifiée. La simplification se traduit ici par l'absence d'algorithme préalable de vérification sur la cardinalité de la classe d'attributs composants et par l'ajout d'une seule contrainte au lieu de deux : si *bloc_moteur* est de type *bloc_pistons*, la cardinalité doit être supérieure ou égale à deux.

Nous avons mentionné dans le chapitre premier que l'état d'une part peut évoluer avec le temps mais que son identité reste toujours la même. Toutefois, si des sous-classes de parts sont associées à des classes d'identités différentes, lorsque l'instance de la super classe racine change de type (donc d'appartenance à une sous-classe feuille), elle modifie automatiquement son identité. Pour éviter cette contradiction, nous associons uniquement les classes d'identités à la super classe racine au sein d'une hiérarchie de spécialisation, les sous-classes héritant automatiquement des classes d'attributs identités de leur super classe racine. La même

restriction est appliquée aux classes de positions dans le but d'empêcher le recours à des contraintes visant à interdire qu'une part soit placée dans des positions différentes lorsqu'elle change de type.

Nous avons introduit dans le sous-ensemble une autre modification très importante par rapport au pattern *Part* : nous avons interdit le principe de l'héritage multiple. Ce dernier pose en effet de nombreux problèmes dans les langages orientés objets. Dans notre cas, il complexifie la gestion des contraintes tant au niveau de la modélisation graphique des "parts" qu'au niveau du langage AlbertII. Rappelons à ce sujet qu'une classe de parts hérite des classes d'attributs de ses super classes et notamment des classes d'attributs identités et positions de sa super classe racine. Or, dans l'éventualité d'un héritage multiple, plusieurs super classes racines coexistent. Si au moins deux d'entre elles sont associées à une classe de positions, cela pose problème car une part ne peut posséder qu'une seule position. Une série de contraintes rédigées en AlbertII doivent donc garantir l'unicité de la position de chaque instance des classes de parts concernées par l'héritage multiple. Celui-ci rend également difficile la transposition en AlbertII des contraintes relatives aux super classes racines, notamment les contraintes d'énumération, dont les types (feuilles) héritent d'autres super classes racines. Par ailleurs, la traduction d'une classe de parts qui est le sous-ensemble de plusieurs classes n'est pas prévue par la syntaxe AlbertII.

En conclusion, l'héritage multiple contraint le programmeur à collecter des informations éparses dans le graphe des hiérarchies de spécialisation et à prendre en considération un nombre considérable de configurations possibles. Il rend aussi la construction du modèle graphique de "parts" plus aléatoire pour l'utilisateur et sa compréhension plus difficile pour le lecteur. Son interdiction résulte donc d'une simplification mais ne constitue en aucun cas une restriction essentielle par rapport au pattern *Part* parce que les modèles graphiques de "parts" qui font intervenir des héritages multiples peuvent être transformés sans perte de sens en modèles constitués exclusivement d'héritages simples.

La nouvelle cardinalité de la méta-classe d'attributs sous-classes ("0:1") traduit le principe qu'une classe de parts ne peut être le sous-ensemble que d'au plus une autre classe de parts.

Dans les interactions entre les relations composants et sous-ensembles, nous partons du principe que, lorsqu'une classe de parts composées est associée par l'intermédiaire d'une classe d'attributs composants à une classe de parts qui possède un ou plusieurs sous-ensembles, une instance de cette classe de parts composées peut être constituée d'instances de cette super classe et de n'importe laquelle de ses sous-classes.

Les contraintes de cardinalité sur les classes d'attributs sont exprimées sur le graphe de la figure 3.3 sous la forme de cardinalités minimales et maximales. Les contraintes sur les méta-classes conservent le même sens que celles présentées au premier chapitre. Toutefois, seules la troisième et la sixième contraintes n'ont pas été modifiées pour être adaptées aux changements opérés par rapport au pattern *Part* :

1. La méta-classe de parts mixtes (*BasicOrCompoundPartClass*), la méta-classe de parts de base (*BasicPartClass*) et la méta-classe de parts composées (*CompoundPartClass*) forment une partition de la méta-classe de parts (*PartClass*). Cela signifie qu'une classe de parts ne peut être simultanément une classe de parts mixtes, une classe de parts de base et une classe de parts composées ou deux de ces trois classes mais qu'elle doit être l'une des trois. Dans la thèse, la description du pattern *Part* se limitait au caractère exclusif des méta-classes de parts de base et de parts composées. Mais suite à la disparition de la méta-classe des parts existantes, ces deux méta-classes forment avec la nouvelle méta-classe de parts mixtes une partition.

2. Les méta-classes de dispositifs géométriques (*PartGeometricalFeatureClass*), physiques (*PartPhysicalFeatureClass*) et de fixation (*PartFixingFeatureClass*) forment également une partition de la méta-classe de dispositifs (*PartFeatureClass*).
3. Si une occurrence de la méta-classe de parts composées (*CompoundPartClass*) est définie sur la base d'une (ou de plusieurs) occurrences de la méta-classe de parts (*PartClass*) – comme ses composants possibles – et d'une (ou de plusieurs) occurrences de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*), alors ces occurrences de la méta-classe de dispositifs géométriques doivent être définies sur la base d'un sous-ensemble des instances de la méta-classe de parts qui sont utilisées (comme composants possibles) pour définir cette occurrence de la méta-classe de parts composées.
4. Si une occurrence de la méta-classe de dispositifs de fixation (*PartFixingFeatureClass*) est définie sur la base d'une (ou de plusieurs) occurrences de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*), alors ces occurrences doivent être définies sur la base d'une (ou de plusieurs) occurrences de la méta-classe de parts (*PartClass*) qui forment un sous-ensemble des occurrences de la méta-classe de parts qui définissent une (même) occurrence de la méta-classe de parts composées (*CompoundPartClass*) comme ses composants possibles.
5. Il existe tout au plus une instance de (chaque) méta-classe d'attributs parmi deux instances de méta-classe de parts ou d'états.
6. Les instances de la méta-classe d'attributs sous-classes (*subclass*) forment un ordre partiel sur les classes de parts.

3.1.1.3 Les classes spéciales

Au premier chapitre, nous avons présenté la classe des tampons et la classe des piles de parts qui sont les deux classes spéciales de parts composées décrites dans la thèse. Nous allons maintenant définir la troisième classe spéciale de parts composées, celle qui se rapporte aux conteneurs. Pour rappel, les conteneurs renferment des piles. Dans la classe des piles de parts dans un conteneur (*PileOfPartsInContainer*), un conteneur est contigu à toutes les parts de la pile et les enclot de telle manière qu'il est impossible d'en enlever une seule, à l'exception de celles situées à l'une ou aux deux extrémités selon que le conteneur est de type FIFO, LIFO ou standard (pile ouverte aux deux extrémités).

Pour définir la classe des conteneurs, nous devons dépasser le cas particulier de la pile et généraliser les caractéristiques de contiguïté, d'enclosure et de contenance pour un ensemble de parts qui sont contenues dans un conteneur ou qui sont les composants d'une ou de plusieurs parts composées qui sont contenues dans ce conteneur et ce, quelle que soit la position de ces parts. Prenons trois exemples différents empruntés à la construction automobile. Soit les parts de base châssis et boîte de vitesse. Soit la part composée moteur constituée de quatre pistons et de deux moitiés de bloc (bloc gauche et bloc droit) formant l'enveloppe extérieure du moteur. Dans le premier exemple, les deux blocs du moteur sont contigus et enclosent les pistons. Le conteneur contient le moteur et les trois classes spéciales de dispositifs géométriques s'exercent sur ses composants. Dans le deuxième exemple, le bloc gauche du moteur et le châssis enclosent la boîte de vitesse. Le conteneur contient les trois parts principales et les trois classes spéciales de dispositifs géométriques s'exercent sur un composant du moteur et sur les deux parts de base. Dans le troisième exemple, le moteur et le châssis enclosent la boîte de vitesse. Le conteneur contient ici aussi les trois parts principales sur lesquelles s'exercent les trois classes spéciales de dispositifs géométriques. En conclusion, le conteneur peut contenir une ou plusieurs parts et peut exercer des contraintes d'accessibilité soit sur des parts composantes, soit sur des parts composées ou des parts de base, soit sur un mélange des deux.

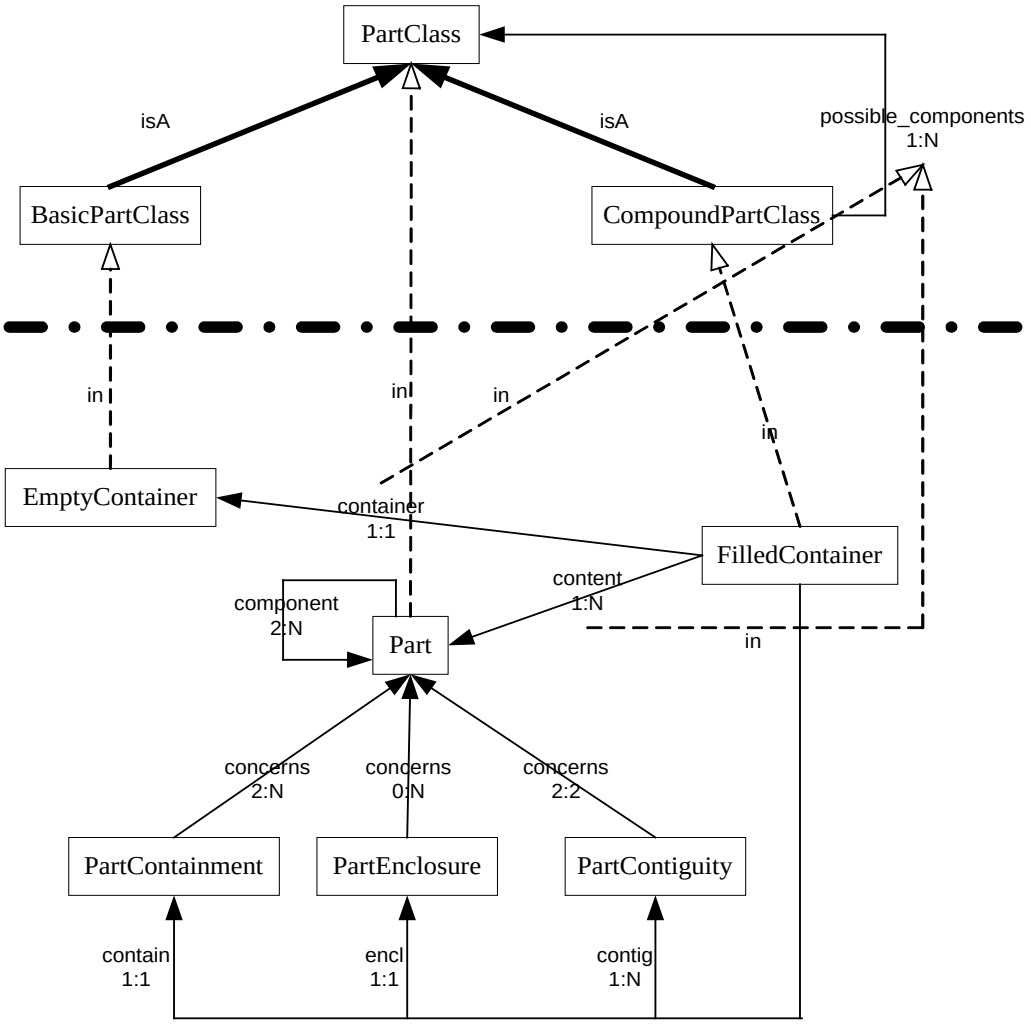


Figure 3.4 : Description graphique de la classe des conteneurs du pattern *Part*

Le graphe de la figure 3.4 montre les classes nécessaires à la définition des conteneurs. La partie supérieure du graphe délimitée par l'épaisse ligne pointillée horizontale correspond aux méta-classes du pattern *Part* tandis que la partie inférieure correspond aux classes spéciales de parts et de dispositifs géométriques. Les flèches en pointillés caractérisent les liaisons d'instanciation entre les deux parties. La borne inférieure de la cardinalité de la classe d'attributs concernés dont l'origine est la classe d'enclos est nulle car un conteneur peut n'enclore aucune part. Celle dont l'origine est la classe de contenants est égale à deux car le conteneur exerce ses contraintes d'accessibilité sur au moins deux parts (qui sont contiguës comme l'indique la cardinalité de la classe d'attributs contiguïtés).

La modélisation particulière de la classe des conteneurs (voir la figure ci-dessus) et de la classe des tampons (voir la figure 1.4) est guidée par la contrainte de composition qui traduit la cardinalité de la classe générale d'attributs composants : une part composée est toujours constituée d'au moins deux parts. Toutefois, à l'inverse des autres parts composées qui n'existent que si elles possèdent plusieurs composants, le conteneur ou le tampon continue d'exister même lorsqu'il est vide. D'où l'idée, selon nous, de faire du conteneur ou du tampon vide un composant obligatoire du conteneur ou du tampon rempli : si le conteneur ou le

tampon ne possède pas de composants, le conteneur ou le tampon vide subsiste seul comme part de base. Par contre, s'il contient au moins un composant en plus de l'élément vide obligatoire, il est représenté par le conteneur ou le tampon rempli comme part composée.

La particularité de la classe de conteneurs et de la double relation contenant et contenu nous incite cependant à réfléchir sur la fonction du conteneur dans notre représentation des systèmes manufacturiers. Si l'on s'en tient à la signification de la part composée, un conteneur est un objet immatériel. Toutes les parts composées sont immatérielles car elles n'existent qu'au travers de leurs composants. Prenons l'exemple d'une pile de pièces de monnaie : si l'on retire les pièces, la pile n'existe plus. Au regard de ce principe, le sens commun du mot conteneur est trompeur parce qu'il fait référence à une caisse (un élément matériel) pouvant contenir des marchandises en vrac.

Nous pouvons pousser le raisonnement plus loin en estimant que l'unique raison d'être du conteneur dans la thèse consiste à définir une propriété de la pile selon laquelle toutes les parts situées à l'intérieur de cette pile sont encloses de telle manière qu'il est impossible d'enlever l'une d'entre elles, à l'exception de celles situées à l'une ou aux deux extrémités. Dans ce cas, le conteneur, qui représente une caractéristique de la pile plutôt qu'un objet contenant une pile, n'est pas une véritable part composée : il détermine simplement si la pile est uniquement ouverte à l'une ou aux deux extrémités. Nous pouvons tenir le même raisonnement si nous généralisons la définition du conteneur comme étant un type de parts pouvant renfermer n'importe quel autre type de parts sur lesquelles s'exercent des contraintes d'accessibilité. Les trois types de dispositifs qui définissent le conteneur n'apportent pas d'informations supplémentaires : ils décrivent qu'un ensemble de parts sont si près les unes des autres qu'il n'est pas possible d'ajouter un composant entre ces parts (contiguïté), qu'elles empêchent collectivement l'accès à un ensemble d'autres parts (enclosure) et qu'elles sont contenues dans une seule part (le conteneur) sans être nécessairement encloses par cette part (contenance).

Toutefois, le modèle de la classe de conteneurs décrit ci-dessus (figure 3.4) et construit à partir du modèle de la classe d'une pile de parts dans un conteneur présenté au premier chapitre (figure 1.6) contient une classe de conteneurs vides (*EmptyContainer*). Un conteneur existerait-il par lui-même en dehors de ses contenus ? Nous ne le pensons pas. Nous supposons que l'existence de la classe de conteneurs vides est due au respect de la définition de la part composée et de la cardinalité de la classe générale d'attributs composants qui exigent qu'une part composée soit toujours constituée d'au moins deux composants : en incluant automatiquement un conteneur vide, le conteneur rempli peut en effet ne contenir qu'une seule autre part comme dans le premier exemple relatif au moteur.

La modélisation des classes spéciales de parts composées et de dispositifs géométriques au niveau du sous-ensemble du pattern *Part* dans la perspective d'une simplification et d'une intégration au logiciel Visio, grâce à une représentation adaptée à l'interface graphique, provoque des contradictions entre les contraintes sur les classes et les définitions sur les parts. En voici quelques unes dont nous devons tenir compte dans la modélisation des classes spéciales :

- une part composée se définit comme un agrégat formé d'un ensemble d'autres parts alors qu'un tampon peut être vide (un conteneur aussi si l'on s'en tient à la modélisation définie dans la thèse);
- en corrélation avec cette définition, une part composée n'existe qu'au travers de ses composants (une pile vide n'existe pas) alors que, conceptuellement, le tampon existe par lui-même, et que, en pratique, vu sa modélisation dans la thèse, le conteneur aussi;
- contrairement au tampon, la part composée "standard", la pile et le conteneur nécessitent l'utilisation de dispositifs géométriques (spéciaux);

- la contrainte de cardinalité de la classe générale d'attributs composants héritée par toutes les classes de parts composées exige que chaque part composée soit constituée d'au moins deux parts alors qu'un tampon peut ne contenir aucune part (un conteneur aussi si l'on s'en tient à la modélisation définie dans la thèse);
- la septième contrainte sur les classes générales stipule qu'aucune part ne peut être le composant de deux parts composées alors qu'une part peut être à la fois le composant d'une part composée et le contenu d'un conteneur puisque ce dernier ne représente pas un objet concret mais plutôt des contraintes d'accessibilité sur les parts qu'il contient. On peut ainsi imaginer une voiture notamment constituée d'un moteur qui est contenu dans un conteneur indiquant que les pistons de ce moteur sont enclos par les deux blocs du moteur.

Si la pile et le conteneur peuvent être considérés comme des parts (composées) parce qu'ils représentent des objets physiques (dans la thèse !) destinés à être développés et produits par les systèmes manufacturiers, ce n'est pas le cas du tampon qui est défini comme un endroit où un certain nombre de parts peuvent demeurer quelque temps. Toutefois, l'assimilation du tampon à une part permet de le modéliser au moyen du logiciel Visio puis de le traduire dans la partie déclaration des types de données du langage AlbertII. Mais puisque le tampon existe par lui-même, c'est-à-dire en dehors de tout contenu, et puisqu'il ne requiert pas l'utilisation de dispositifs géométriques (spéciaux), nous avons décidé de le modéliser comme une méta-classe de parts au même titre que les méta-classes de parts de base, de parts composées et de parts mixtes. La nouvelle méta-classe de tampons (*BufferClass*) devient ainsi une spécialisation de la méta-classe de parts (*PartClass*) à qui nous attribuons une méta-classe d'attributs contenus (*possible_contents*). Nous créons également une classe générale de tampons (*Buffer*) et une classe générale d'attributs contenus (*content*) correspondantes.

Dans les schémas suivants, nous avons modifié le nom de la plupart des classes spéciales de parts, de dispositifs et d'attributs afin de les différencier, de les simplifier et de marquer la filiation de certaines d'entre elles avec la classe de dispositifs géométriques grâce à l'ajout du terme "dispositif" ("*feature*").

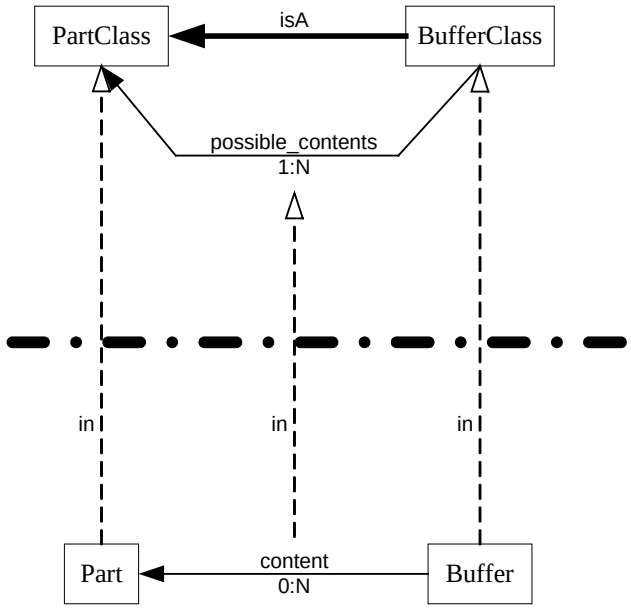


Figure 3.5 : Description graphique de la méta-classe des tampons (*BufferClass*) et de la classe générale des tampons (*Buffer*) du sous-ensemble du pattern *Part*

La figure 3.5 présente le graphe de la classe des tampons du sous-ensemble du pattern *Part*. La suppression de la classe spéciale des tampons vides (*EmptyBuffer*) et de la classe spéciale d'attributs contenants (*container*) dont elle est la destination rend inutiles les première et troisième contraintes sur les tampons. Désormais, un tampon vide n'est plus une part de base spéciale mais bien une part spéciale qui fonctionne comme une part de base parce qu'elle n'est constituée d'aucun composant. Toutefois, la classe de tampons doit être associée à au moins une classe de parts soit directement soit par héritage car un tampon perpétuellement vide ne servirait à rien.

Au sein du sous-ensemble du pattern *Part*, nous conservons la définition de la classe de piles de parts (*PileOfParts*) donné dans la thèse qui la présente comme une instance de la méta-classe de parts composées (*CompoundPartClass*) et un sous-ensemble de la classe générale de parts composées (*CompoundPart*) qui contient comme composants des instances de la méta-classe de parts (*PartClass*) et qui possède un certain nombre de dispositifs géométriques dérivés de la classe de contiguïté (*PartContiguity*). Cette définition ne fait nullement référence au conteneur. En outre, d'après la modélisation de la classe d'une pile de parts dans un conteneur (*PileOfPartsInContainer*), qui est une spécialisation de l'ancienne classe des conteneurs remplis (*FilledContainer*), le conteneur associé aux différents dispositifs géométriques spéciaux ne contient qu'une et une seule pile. La classe des piles se substitue donc facilement à la classe d'une pile de parts dans un conteneur. Elle hérite par conséquent des différentes classes spéciales de dispositifs géométriques.

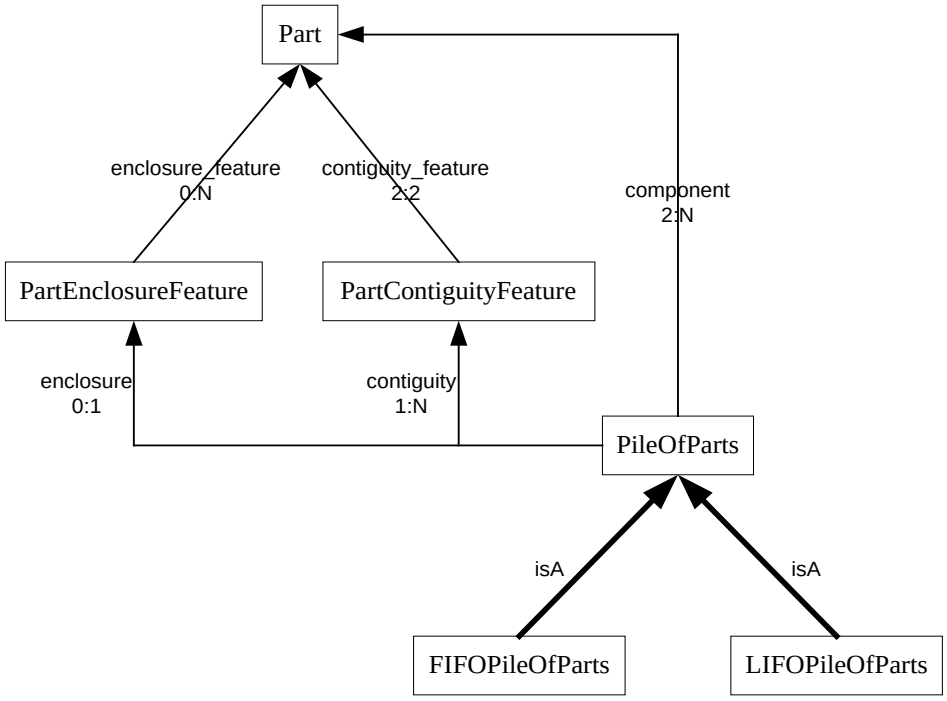


Figure 3.6 : Description graphique de la classe des piles (*PileOfParts*) du sous-ensemble du pattern *Part*

La figure 3.6 présente le graphe de la classe des piles (*PileOfParts*) du sous-ensemble. Nous avons modifié le modèle de cette classe défini dans le pattern *Part* pour le simplifier et l'adapter à la modélisation graphique des "parts". En effet, puisque la pile exerce elle-même les contraintes d'accessibilité sur ses composants directs, la classe générale d'attributs composants (*component*) se substitue à la classe spéciale d'attributs contenus (*content*) et la classe spéciale d'attributs contenants (*contain*) n'est plus d'aucune utilité. Comme décrit dans la thèse, les éléments d'une pile restent placés les uns sur les autres et peuvent toujours être enclos de telle manière que seuls ceux situés à l'une ou aux deux extrémités de la pile puissent être enlevés. La nouvelle classe de piles FIFO (*FIFOPileOfParts*) et la nouvelle classe de piles LIFO (*LIFOPileOfParts*) remplacent les anciennes classes de conteneurs FIFO et LIFO et deviennent des sous-ensembles de la classe des piles. Elles constituent des sous-classes exclusives mais ne forment pas une partition de la classe des piles car celle-ci n'est pas une classe abstraite.

Nous avons pris le parti d'employer le conteneur comme une entité virtuelle qui définit les propriétés de contiguïté et d'enclosure qui s'exercent sur les parts qu'elle contient. Dans le sous-ensemble du pattern *Part*, nous allons considérer la classe des conteneurs (*Container*) comme une classe spéciale de parts composées qui est une instance de la méta-classe de parts composées (*CompoundPartClass*) et un sous-ensemble de la classe générale de parts composées (*CompoundPart*). Cela permet à la classe de conteneurs d'hériter des classes spéciales de dispositifs géométriques dont elle a besoin. Elle dispose également de classes d'attributs composants. Toutefois, pour éviter de contrevenir à la septième contrainte sur les classes générales qui stipule qu'aucune part ne peut être le composant de deux parts composées, nous lui attribuons à la place une classe spéciale d'attributs contenants (*contain*) qui est une instance de la méta-classe d'attributs composants (*possible_components*) et une spécialisation de la classe générale d'attributs composants (*component*). Cette classe d'attributs contenants se démarque des relations de composition habituelles : elle indique que les parts sur lesquelles s'exercent les contraintes d'accessibilité sont contenues dans le conteneur et n'en constituent plus les composants. Sa cardinalité ("2:N") est héritée de la classe générale d'attributs composants, ce qui lui permet de respecter la définition de la part composée considérée comme formée de composants (de contenus dans ce cas-ci) sans lesquels elle n'existerait pas. En conférant ainsi conceptuellement au conteneur le statut de part composée spéciale et en le plaçant au même niveau que la pile et la part composée "standard", nous nous écartons un peu de l'image que lui en donne la thèse où il ne semble défini que par rapport à la pile. Toutefois, en pratique, nous réduisons aussi son rôle à celui d'une propriété indiquant quelles sont les parts qui sont contiguës et encloses.

La figure 3.7 présente le graphe de la classe des conteneurs du sous-ensemble qui est basé sur le modèle de la classe des piles défini précédemment. Il a lui aussi été simplifié et adapté à la modélisation graphique des "parts" : selon le même principe que celui appliqué au modèle de la classe des tampons, nous avons supprimé la classe des conteneurs vides (*EmptyContainer*) et la classe d'attributs conteneurs (*container*) dont elle est la destination. Désormais, un conteneur vide n'existe plus. La classe de contenants (*PartContainment*) et les classes d'attributs associées (*contain* et *concerns*) sont avantageusement remplacées par la nouvelle classe d'attributs contenants (*contain*). Quant à l'ancienne classe d'attributs contenus (*content*), elle disparaît parce que, le conteneur étant une entité virtuelle, nous n'avons plus besoin de connaître quelles sont les parts composées dont les composants subissent les contraintes d'accessibilité. Désormais, nous nous limitons à indiquer quelles sont les parts concernées par les propriétés de contiguïté et d'enclosure au moyen de la classe d'attributs contenants.

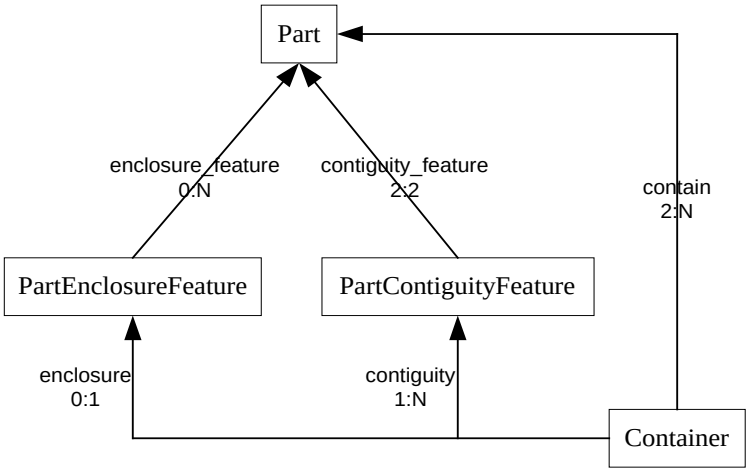


Figure 3.7 : Description graphique de la classe des conteneurs (*Container*) du sous-ensemble du pattern *Part*

Dans le sous-ensemble du pattern *Part*, nous allons considérer les classes de dispositifs de contiguïté (*PartContiguityFeature*) et d'enclos (*PartEnclosureFeature*) comme des classes spéciales de dispositifs géométriques qui sont des instances de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*) et des sous-ensembles de la classe générale de dispositifs géométriques (*PartGeometricalFeature*). Toutes les classes spécifiques de dispositifs d'enclos et de contiguïté seront identifiées comme des sous-classes respectives de ces deux classes spéciales. Ces dernières fonctionneront comme des sous-ensembles exclusifs de la classe générale de dispositifs géométriques dont les instances posséderont des propriétés particulières. Par ailleurs, puisque les dispositifs d'enclos et de contiguïté sont des dispositifs géométriques, ils pourront posséder des dispositifs de valeur mais ne pourront définir des dispositifs de fixation.

Nous allons également considérer les classes d'attributs contiguïtés (*contiguity*) et enclos (*enclosure*) comme des classes spéciales d'attributs géométries qui sont des instances de la méta-classe d'attributs géométries (*possible_geometry*) et des sous-ensembles de la classe générale d'attributs géométries (*geometry*), et les classes d'attributs dispositifs de contiguïté (*contiguity_feature*) et d'enclos (*enclosure_feature*) comme des classes spéciales d'attributs dispositifs géométriques qui sont des instances de la méta-classe d'attributs dispositifs géométriques (*possible_geometrical_features*) et des sous-ensembles de la classe générale d'attributs dispositifs géométriques (*geometrical_feature*). Toutes les classes spécifiques d'attributs contiguïtés, enclos, dispositifs de contiguïté et dispositifs d'enclos seront identifiées comme des sous-classes respectives de ces quatre classes spéciales. Les deux premières fonctionneront comme des sous-ensembles exclusifs de la classe générale d'attributs géométries et les deux suivantes comme des sous-ensembles exclusifs de la classe générale d'attributs dispositifs géométriques dont les instances posséderont des propriétés particulières.

Toutes les classes spécifiques de piles (de parts) et de conteneurs seront reconnues comme des sous-classes respectives de la classe spéciale des piles de parts (*PileOfParts*) et de la classe spéciale des conteneurs (*Container*). Ces dernières formeront des sous-ensembles exclusifs de la classe générale de parts composées (*CompoundPart*) dont les instances posséderont des propriétés particulières. Quant aux classes spécifiques de tampons, elles

seront reconnues comme des instances de la méta-classe de tampons (*BufferClass*) et des sous-ensembles de la classe générale des tampons (*Buffer*). Par ailleurs, puisque les piles, les conteneurs et les tampons sont des parts, ils pourront posséder une identité, une position et des dispositifs physiques, bien que les conteneurs soient considérés comme des entités virtuelles.

Les classes spécifiques d'attributs contenus seront considérées comme des instances de la méta-classe d'attributs contenus (*possible_contents*) et des spécialisations de la classe générale d'attributs contenus (*content*), tandis que les classes spécifiques d'attributs contenants seront définies comme des sous-classes de la classe spéciale d'attributs contenants (*contain*).

Les classes spécifiques de dispositifs de contiguïté et d'enclos étant des spécialisations des classes spéciales du même nom qui sont elles-mêmes des instances de la méta-classe de dispositifs géométriques, elles doivent posséder au moins une classe d'attributs dispositifs de contiguïté et une classe d'attributs dispositifs d'enclos car la borne inférieure de la cardinalité de la méta-classe d'attributs dispositifs géométriques équivaut à un. De la même manière, les classes spécifiques de piles et de conteneurs étant des spécialisations des classes spéciales du même nom qui sont elles-mêmes des instances de la méta-classe de parts composées, elles ne doivent pas nécessairement posséder de classe d'attributs contiguïtés ou enclos car la borne inférieure de la cardinalité de la méta-classe d'attributs géométries équivaut à zéro. Cependant, les éléments d'une pile ou d'un conteneur étant contigus, nous imposons que chaque classe de piles ou de conteneurs détienne au moins une classe d'attributs contiguïtés.

Les cardinalités des classes spéciales d'attributs contiguïtés ("1:N") et enclos ("0:1") respectent celle de la classe générale d'attributs géométries ("0:N") dont ces classes sont des spécialisations en renforçant toutefois les contraintes liées à cette cardinalité. Par contre, si c'est également le cas de la cardinalité de la classe spéciale d'attributs dispositifs de contiguïté ("2:2") vis-à-vis de la cardinalité de la classe générale d'attributs dispositifs géométriques ("2:N"), ce n'est pas le cas de celle de la classe spéciale d'attributs dispositifs d'enclos ("0:N") parce qu'un conteneur ou une pile ouverte uniquement aux deux extrémités peut n'enclore aucune part. Nous tolérons ici une entorse à la règle de l'héritage pour une raison pratique.

Cette modélisation permet d'utiliser un conteneur pour indiquer que des parts sont uniquement contiguës et non pas encloses. Les éléments d'une pile qui ne possède pas de dispositif d'enclos sont tous accessibles. Dans le cas contraire, seuls ceux qui se trouvent à l'une ou aux deux extrémités de la pile sont accessibles.

Dans les interactions entre les relations contenus/contenants et sous-ensembles, nous partons du principe que, lorsqu'une classe de tampons ou de conteneurs est associée par l'intermédiaire d'une classe d'attributs contenus/contenants à une classe de parts qui possède un ou plusieurs sous-ensembles, une instance de cette classe de tampons ou de conteneurs peut contenir des instances de cette super classe et de n'importe laquelle de ses sous-classes.

Ces choix de modélisation se situent à mi-chemin des modèles minimaliste et maximaliste que nous aurions pu adopter l'un comme l'autre. Le premier consiste à supprimer les classes de piles et de conteneurs et à les remplacer par des classes de parts composées. Les propriétés spécifiques aux piles (FIFO, LIFO et ouverture aux deux extrémités) peuvent être indiquées en typant les dispositifs d'enclos ou en utilisant un dispositif physique spécifique. Seules les classes de dispositifs de contiguïté et d'enclos sont conservées. La possibilité d'exercer des contraintes d'accessibilité sur des composants est offerte en utilisant un attribut contenant à la place d'un attribut composant. Une part composée peut ainsi devenir un composé, un contenant ou les deux à la fois. Cela réduit le nombre de types de parts à quatre.

Dans la configuration maximaliste, l'ensemble des classes spéciales (d'attributs) telles qu'elles sont présentées dans la thèse et à la figure 3.4 sont transformées en méta-classes (d'attributs) et se comportent comme les méta-classes (d'attributs) du pattern *Part*. Les méta-classes de piles de parts, de conteneurs remplis et de tampons remplis deviennent des sous-ensembles de la méta-classe de parts, tandis que les méta-classes de conteneurs vides et de tampons vides deviennent des sous-ensembles de la méta-classe de parts de base. Les méta-classes de dispositifs de contiguïté, d'enclos et de contenant deviennent des sous-ensembles de la méta-classe de dispositifs. Quant aux nouvelles méta-classes d'attributs, elles acquièrent un statut autonome. Des classes générales (d'attributs) correspondantes sont également créées.

Dans notre modèle, six types de classes spécifiques de parts coexistent : la classe de parts de base (*BasicPart*), la classe de parts mixtes (*BasicOrCompoundPart*), la classe de piles de parts (*PileOfParts*), la classe de conteneurs (*Container*), la classe de tampons (*Buffer*) et la classe de parts composées "standards" (*CompoundPart*) dont les instances sont des parts composées qui ne sont ni des piles, ni des conteneurs, ni des tampons. Cette situation vient compliquer la spécialisation des parts car, à l'exception des classes de parts mixtes, les classes de parts ne peuvent être les super classes que des classes de parts du même type qu'elles parce qu'elles sont toutes des classes exclusives. Une part doit en effet être soit une part de base, soit une part composée "standard", soit une pile, soit un conteneur, soit un tampon, et ne peut être deux de ces types à la fois. Ce n'est pas le cas de la part mixte qui n'existe pas en elle-même et qui doit donc être en même temps un autre type sauf un tampon ou un conteneur.

Ces restrictions sont logiques au regard du principe d'héritage au sein des hiérarchies de spécialisation. Par exemple, une pile ne peut hériter d'un attribut contenu ou contenant, un conteneur ne peut hériter d'un attribut composant ou contenu et seul un tampon peut hériter d'une position fixe. L'usage des classes spécifiques de parts mixtes dans les modèles graphiques de "parts" doit donc être redéfini : si nous représentons la hiérarchie des sous-ensembles comme un arbre, une classe de parts mixtes ne peut être qu'une super classe qui possède au moins, soit une sous-classe directe de parts mixtes, soit deux sous-classes directes de parts appartenant à des types différents et autres que le tampon et le conteneur.

Le tableau 3.1 résume les spécialisations possibles entre les classes de parts. Pour chaque ligne du tableau, un booléen indique si le type de classes de parts peut être la super classe d'un type de classes de parts situé dans une des colonnes. Inversement, pour chaque colonne du tableau, un booléen indique si le type de classes de parts peut être la sous-classe d'un type de classes de parts placé dans une des lignes.

est la super classe de	**Part de base**	**Part composée**	**Part mixte**	**Pile**	**Conteneur**	**Tampon**
Part de base	V	F	F	F	F	F
Part composée	F	V	F	F	F	F
Part mixte	V	V	V	V	F	F
Pile	F	F	F	V	F	F
Conteneur	F	F	F	F	V	F
Tampon	F	F	F	F	F	V

Tableau 3.1 : Spécialisation des classes de parts

Nous avons écrit au premier chapitre que la thèse permettait de déclarer des classes de tampons comme des spécialisations des classes de piles de parts afin de pouvoir définir des classes de tampons LIFO. Notre modélisation ne le permet pas. Mais nous pouvons créer des tampons LIFO en déclarant une classe de tampons qui contient une classe de piles LIFO qui ne peut être instanciée qu'une seule fois et qui est constituée de toutes les classes de parts que nous souhaitons mettre dans les tampons LIFO. Toutefois, un problème subsiste si le tampon doit contenir une seule part car une pile ne peut être composée d'une seule part et, par conséquent, en l'absence de pile, le tampon reste vide. La solution consiste à associer la classe de tampons à chaque classe de parts composantes au moyen d'une classe d'attributs contenus de cardinalité "1:1". Ainsi, la relation qui unit le tampon à la part unique est symbolisée par un attribut contenu. Comme nous l'avons déjà fait remarquer, une part seule ne nécessite pas l'utilisation de dispositifs de contiguïté et d'enclos. Dans tous les autres cas, la relation s'effectue par l'intermédiaire de la pile. Cependant, une contrainte rédigée en AlbertII doit exprimer cette configuration particulière.

Suite à l'apparition d'une nouvelle méta-classe et d'une nouvelle classe générale de parts, les premières contraintes relatives aux méta-classes et classes générales doivent être adaptées :

- les classes générales de parts de base (*BasicPart*), de tampons (*Buffer*) et de parts composées (*CompoundPart*) forment une partition de la classe générale des parts (*Part*);
- les méta-classes de parts mixtes (*BasicOrCompoundPartClass*), de tampons (*BufferClass*), de parts de base (*BasicPartClass*) et de parts composées (*CompoundPartClass*) forment une partition de la méta-classe de parts (*PartClass*).

Les contraintes de cardinalité sur les attributs sont exprimées sur les graphes des figures 3.5 à 3.7 sous la forme de cardinalités minimales et maximales. Les quatre premières contraintes sur les classes spéciales de piles de parts, de conteneurs et de tampons sont héritées de celles présentées au premier chapitre. Les autres sont des nouvelles contraintes apparues suite à l'adaptation des modèles de classes spéciales au sous-ensemble :

1. Dans une classe de piles de parts (*PileOfParts*), chaque composant est contigu à exactement deux autres composants, à l'exception de deux d'entre eux (localisés aux extrémités) qui sont contigus à un seul composant.
2. Dans une classe de piles FIFO (*FIFOPileOfParts*), LIFO (*LIFOPileOfParts*) ou ouverte aux deux extrémités (*PileOfParts* associée à une classe de dispositifs d'enclos), une pile enclot tous ses composants de telle manière qu'il est impossible d'enlever l'un d'entre eux, à l'exception de ceux situés à l'une ou aux deux extrémités.
3. Dans une classe de piles (*PileOfParts*), les contraintes d'accessibilité (contiguïté et enclosure) de la pile s'exercent sur les parts qui composent directement cette pile. Dans une classe de conteneurs (*Container*), les contraintes d'accessibilité (contiguïté et enclosure) du conteneur s'exercent sur les parts qui sont directement contenues dans ce conteneur. En pratique, chaque dispositif de contiguïté concerne (accole) deux composants directs d'une même pile ou deux contenus directs d'un même conteneur. Quant au dispositif d'enclos d'une pile ou d'un conteneur, il concerne (enclot) uniquement les composants directs de cette pile ou les contenus directs de ce conteneur.
4. Un tampon possède une position fixe.
5. Aucune pile n'est composée d'elle-même.
6. Aucun tampon ne se contient lui-même. Un conteneur ne peut pas se contenir lui-même car, étant une entité virtuelle, il ne peut pas être un contenu.
7. Aucune part ne peut être contenue à la fois par deux tampons. Aucune part ne peut être à la fois le contenu d'un tampon et le composant d'une part composée.
8. Les classes spéciales de piles FIFO (*FIFOPileOfParts*) et LIFO (*LIFOPileOfParts*) sont des classes exclusives : une pile ne peut appartenir simultanément à deux de ces classes.

9. Les classes spéciales de dispositifs de contiguïté (*PartContiguityFeature*) et d'enclos (*PartEnclosureFeature*) sont des classes exclusives : un dispositif géométrique ne peut appartenir simultanément à deux de ces classes.
10. Les classes spéciales d'attributs dispositifs de contiguïté (*contiguity_feature*) et d'enclos (*enclosure_feature*) sont des classes exclusives : un attribut dispositif géométrique ne peut appartenir simultanément à deux de ces classes.
11. Les classes spéciales d'attributs contiguïtés (*contiguity*) et enclos (*enclosure*) sont des classes exclusives : un attribut géométrie ne peut appartenir simultanément à deux de ces classes.
12. Si une spécialisation de la classe spéciale de piles de parts (*PileOfParts*) ou de conteneurs (*Container*) est définie sur la base d'une (ou de plusieurs) occurrences de la méta-classe de parts (*PartClass*) – comme ses composants ou ses contenus possibles – et d'une (ou de plusieurs) spécialisations de la classe spéciale de dispositifs de contiguïté (*PartContiguityFeature*), alors ces spécialisations de la classe spéciale de dispositifs de contiguïté doivent être définies sur la base des instances de la méta-classe de parts qui sont utilisées (comme composants possibles) pour définir la classe de piles ou sur la base d'un sous-ensemble des instances de la méta-classe de parts qui sont utilisées (comme contenus possibles) pour définir la classe de conteneurs.
13. Si une spécialisation de la classe spéciale de piles de parts (*PileOfParts*) ou de conteneurs (*Container*) est définie sur la base d'une (ou de plusieurs) occurrences de la méta-classe de parts (*PartClass*) – comme ses composants ou ses contenus possibles – et d'une spécialisation de la classe spéciale de dispositifs d'enclos (*PartEnclosureFeature*), alors cette spécialisation doit être définie sur la base d'un sous-ensemble des instances de la méta-classe de parts qui sont utilisées (comme composants ou contenus possibles) pour définir la classe de piles ou de conteneurs.
14. Les classes spéciales de piles de parts (*PileOfParts*) et de conteneurs (*Container*) sont des classes exclusives : une part composée ne peut appartenir simultanément à deux de ces classes.

L'annexe A présente l'intégralité du méta-modèle du sous-ensemble du pattern *Part*.[16] La partie centrale du graphe délimitée par l'épaisse ligne pointillée contient les méta-classes (en rouge) tandis qu'à l'extérieur se trouvent les classes générales (en bleu) et les classes spéciales (en jaune). Les flèches en pointillés représentent les liaisons d'instanciation ("in") et les flèches au trait uni plus épais les liaisons de spécialisation ("isA"). Pour éviter d'alourdir le schéma, nous avons omis les relations de spécialisation lorsqu'une super classe et sa sous-classe sont des instances de la même méta-classe. Ceci ne concerne que les classes spéciales. Contrairement à ce qu'indique le graphe, la classe générale d'attributs composants n'est pas héritée par la classe spéciale de conteneurs : la classe spéciale d'attributs contenants se substitue à elle. D'autre part, la configuration du schéma nécessite la présence de la classe générale de parts à deux endroits différents.

Dorénavant, lorsque nous désignerons les nouvelles classes de parts composées, de dispositifs géométriques et d'attributs qui servent de super classes à la modélisation graphique des "parts", nous emploierons le terme de "classes spéciales" et lorsque nous parlerons des instances de ces nouvelles classes, nous utiliserons le vocable de "classes spécifiques" ou, plus simplement, de "classes". Pour désigner une part composée de n'importe quel type, nous emploierons l'expression "part composée au sens large" et pour signaler une part composée qui n'est ni une pile, ni un conteneur, nous emploierons le terme de "part composée".

[16] Pour un exemple de modélisation graphique de "parts", voir l'annexe D.

3.1.2. Correspondance entre le sous-ensemble du pattern *Part* et les concepts Visio

Concepts du sous-ensemble du pattern *Part*	**Concepts Visio**
Sous-ensemble du pattern *Part*	Modèle utilisateur (+ Solution Visio)
Méta-classes et classes spéciales (ensemble des)	Gabarit (de document)
Super méta-classe	Calque
Méta-classe ou classe spéciale	Forme de base
Modèle graphique de "parts"	Projet (= document)
Classe spécifique (instance d'une méta-classe)[17] d'attributs de parts ou d'états	Forme (instance d'une forme de base) unidimensionnelle (flèche) bidimensionnelle
Nom (cardinalité, duplication et ordre)	Propriétés personnalisées
Contraintes (ensemble des) et classes générales	Projet VBA
Contrainte ou propriété d'une classe générale	Macro, procédure, fonction, morceau de code

Tableau 3.2 : Correspondance entre les concepts du sous-ensemble du pattern *Part* et les concepts Visio

Le tableau 3.2 résume la correspondance entre les concepts du pattern *Part* et ceux du logiciel Microsoft Visio. La partie supérieure du tableau correspond à l'espace de travail et aux outils mis à disposition de l'utilisateur, ainsi qu'à la conceptualisation du méta-modèle de "parts", tandis que la partie inférieure coïncide davantage avec la manipulation des objets physiques de Visio et avec les aspects pratiques liés à la construction des modèles graphiques de "parts" particuliers.

Au niveau supérieur consacré au méta-modèle et à l'environnement de travail, le sous-ensemble du pattern *Part* défini à la section précédente trouve son correspondant Visio dans le modèle de dessin intitulé "Systèmes manufacturiers" que nous avons conçu pour réaliser les modèles graphiques de "parts". Derrière ce modèle se profile la solution Visio qui offre un cadre de travail indispensable à l'utilisateur : des outils permettant la manipulation du dessin et de l'interface, ainsi qu'un environnement de développement nécessaire à l'exécution des macros, comprenant notamment Automation et ses bibliothèques.

Le modèle "Systèmes manufacturiers" comprend une série de macros et deux gabarits portant le nom de "Gabarit_SysMan" et de "Gabarit_Pile_Conteneur_Tampon" et renfermant un ensemble de formes de base qui correspondent respectivement aux méta-classes et aux classes spéciales du pattern *Part*. Les formes de base possèdent chacune un formatage particulier qui permet de les distinguer aisément.[18] Toutefois, des formes apparentées présentent des propriétés communes. Par exemple, une couleur identique est attribuée aux deux formes de base qui correspondent, l'une, à une méta-classe d'états et, l'autre, à la méta-classe d'attributs qui associe cette méta-classe d'états à la méta-classe de parts. Autre exemple, les formes des méta-classes de parts de base, composées et mixtes sont

[17] Certaines classes spécifiques sont également le sous-ensemble d'une classe spéciale.

[18] Voir la présentation des formes de base du modèle de dessin "Systèmes manufacturiers" à l'annexe B.

rectangulaires et leur couleur de remplissage est un dégradé de gris-blanc. Le premier gabarit comprend une forme supplémentaire destinée à recueillir les commentaires de l'utilisateur vis-à-vis des classes spécifiques de parts et d'états.

Chaque forme de base est associée à un calque qui groupe les méta-classes et les classes spéciales apparentées. Initialement, les calques devaient représenter uniquement les super méta-classes de parts (*PartClass*) et de dispositifs (*PartFeatureClass*). Mais comme nous souhaitions utiliser les objets calques comme propriétés des formes dans le projet VBA au niveau des macros, nous avons dû assigner un calque à toutes les formes de base sous peine d'entraîner des erreurs d'exécution. Nous avons donc conçu un deuxième type de calques dit "thématique" pour les classes spéciales et les méta-classes d'états (*PartStateClass*), les classes spéciales et les méta-classes d'attributs (*AttributeClass* et *SubClass* qui est réservé à la méta-classe d'attributs sous-classes en raison de son comportement particulier) et les commentaires (*CommentClass*).

Au niveau inférieur, plus pratique, la réalisation d'un modèle graphique de "parts" est assimilée à celle d'un projet de dessin(s) Visio, lui-même qualifié techniquement de "document". Nous pouvons faire un parallélisme entre une classe spécifique qui est une instance d'une méta-classe et une forme sur la page de dessin qui est une instance d'une forme de base.[19] Chaque instance hérite de la mise en forme et des propriétés de la méta-classe/forme de base correspondante. Idéalement, le geste de création d'une forme/classe spécifique marque cette filiation car la technique utilisée consiste à faire glisser la forme de base hors du gabarit pour la placer sur le dessin. En pratique, une seconde technique, la duplication, est également employée : elle consiste à copier une forme déjà présente sur le dessin puis à la coller ailleurs dans le projet. Certaines manipulations supplémentaires doivent généralement accompagner cette méthode comme, par exemple, attribuer un nouveau nom à la forme/classe créée. Le lien avec la forme de base/méta-classe d'origine est conservé. C'est pour maintenir cette liaison qu'une forme ne doit jamais être dessinée à main levée.

Les calques ne peuvent pas être instanciés sur le dessin comme les formes de base. Ils représentent donc des super classes abstraites.

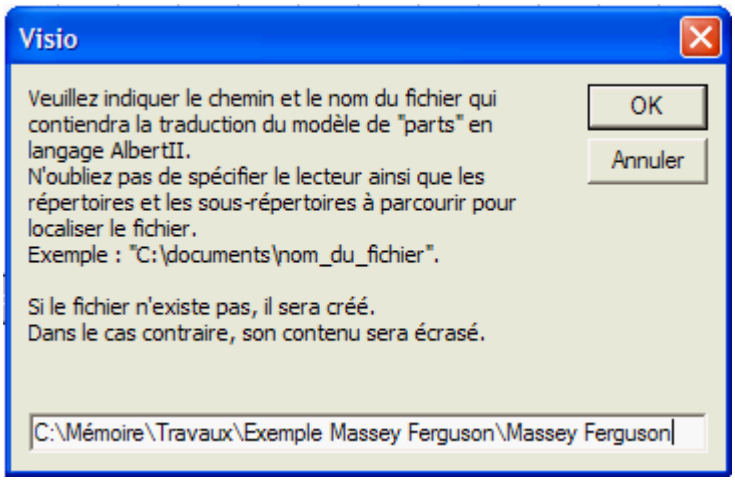


Figure 3.8 : Exemple de boîte de dialogue pour l'enregistrement

[19] Une classe spécifique qui est un sous-ensemble d'une classe spéciale reste toujours une instance d'une méta-classe.

de la traduction du modèle "Exemple Massey Ferguson"

Un modèle graphique de "parts" comprend la définition d'une ou de plusieurs classes de parts autonomes et peut s'étaler sur plusieurs pages d'un même projet.[20] Cela permet d'accroître sa lisibilité. Toutefois, un modèle ne peut être partagé entre plusieurs projets. Il doit donc être complet au moment de sa traduction en AlbertII. Celle-ci peut être déclenchée au moyen d'un bouton placé sur la barre d'outils. Ce bouton exécute une macro qui traduit l'entièreté du modèle graphique de "parts" et qui place le code généré dans un fichier sous format texte (".txt") dont le nom et la localisation peuvent être déterminés par l'utilisateur. Le nom par défaut est "Albert" et la racine par défaut est "C:\". Voir la figure 3.8 dont la boîte d'enregistrement est empruntée à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D.

Le gabarit de document qui contient l'inventaire des formes de base utilisées dans un projet constitue dans notre cas particulier un sous-ensemble du gabarit car toutes les formes de base employées sont celles de ce gabarit et aucune d'entre elles ne peut être modifiée ou supprimée.[21]

Les formes unidimensionnelles (flèches) correspondent aux (méta-)classes d'attributs, tandis que les formes bidimensionnelles correspondent aux (méta-)classes de parts et d'états. À chaque niveau, les premières servent à créer des liaisons entre les secondes. Les classes d'attributs (ou relations) sont représentées par une flèche délimitée par un point de début et un point de fin. La forme collée sur le point de début est appelée l'origine de la relation et la forme collée sur le point de fin est appelée la destination de la relation.

Le nom porté par les classes d'identités, les classes de positions et les classes de dispositifs de valeur est défini par l'utilisateur ou est emprunté aux types de données élémentaires prédéfinis en langage AlbertII : *chaîne de caractères*, *caractère*, *booléen*, *entier*, *rationnel* et *durée*. Dans le premier cas, le nom est aussi un type élémentaire. Ce dernier peut comprendre des sous-types, éventuellement prédéfinis. Bien que cette structure ne puisse pas être décrite dans le modèle graphique de "parts", elle peut faire l'objet d'un commentaire associé à la forme.

Les classes de parts et d'états se différencient des classes d'attributs par la première lettre de leur nom qui est toujours une majuscule.

À deux exceptions près, chaque forme possède une ou plusieurs propriétés personnalisées qui stockent de l'information sur les classes spécifiques : le nom de la classe pour les classes de parts et d'états, la cardinalité des relations (voire la duplication et l'ordre) en plus pour les classes d'attributs.[22] Les formes représentant une classe de piles indiquent également le type de piles : FIFO, LIFO ou standard (piles ouvertes aux deux extrémités). Lorsqu'une forme est glissée ou copiée sur la page de dessin, une boîte de dialogue apparaît pour que l'utilisateur puisse encoder les données nécessaires. Voir en exemple la figure 3.9. Ces données peuvent être modifiées à tout moment grâce à la fenêtre "Fenêtre Propriétés Personnalisées" présente en permanence au niveau de l'interface. Voir la figure 3.10 dont l'exemple est emprunté à l'exemple de modélisation du tracteur Massey Ferguson exposé à l'annexe D. L'utilisateur doit simplement sélectionner la forme correspondant à la classe dont il souhaite changer les données pour que celles-ci apparaissent dans la fenêtre.

---

[20] Deux boutons visibles sur la barre d'outils permettent d'ajouter et de supprimer des pages.

[21] Les deux techniques employées pour créer des formes (glisser des formes de base d'un gabarit et dupliquer des formes existantes) font de chaque forme du projet une instance d'une forme de base (du gabarit) et toutes ces formes de base sont figées.

[22] Les deux exceptions sont les formes représentant les commentaires et les classes d'attributs sous-classes.

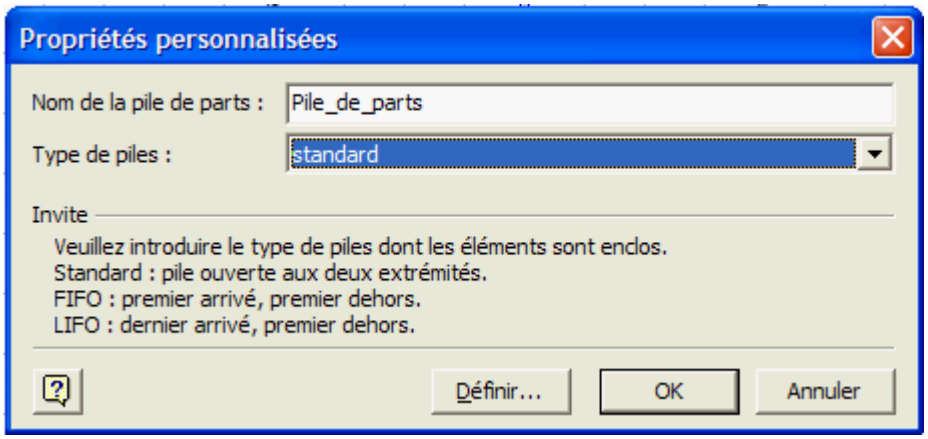


Figure 3.9 : Exemple de boîte de dialogue destinée à enregistrer les données d'une classe de piles après avoir glissé sur le dessin la forme de base la représentant

Chaque forme de base possède aussi trois champs de données dont le premier (*Data1*) contient le nom de la méta-classe ou de la classe spéciale associée à cette forme et le troisième (*Data3*) le numéro qui sert d'identifiant à chaque classe de parts ou d'états. Les formes sur les pages de dessin héritent de ces champs. Nous n'avons pas utilisé une propriété personnalisée pour enregistrer le nom des méta-classes ou des classes spéciales parce que les champs sont plus faciles à manipuler au niveau du code VBA (macros) et parce que, contrairement aux propriétés personnalisées, les données des champs sont fixées et ne doivent pas être modifiées par l'utilisateur.

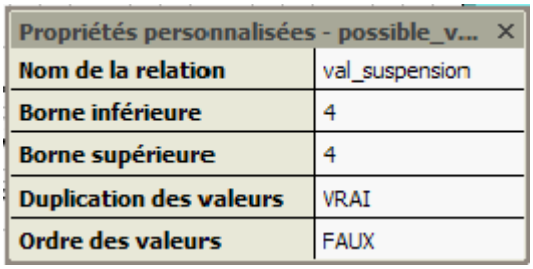


Figure 3.10 : Exemple de "Fenêtre Propriétés Personnalisées" présentant les données de la classe d'attributs dispositifs de valeur *val_suspension*

Chaque forme de base possède une feuille ShapeSheet dont héritent les formes correspondantes créées sur le dessin. Hormis la vérification de certaines contraintes, cette feuille de calcul sert aussi à personnaliser l'aspect et le comportement des formes :

- faire apparaître la boîte de dialogue des propriétés personnalisées lors de la création d'une forme;
- insérer dans le bloc de texte d'une forme le nom de la classe et la cardinalité de la relation qui se trouvent dans les propriétés personnalisées;
- déterminer si le type de piles doit être affiché avec le nom de la classe de piles dans le bloc de texte d'une forme;

- protéger les formes pour empêcher la modification du bloc de texte d'une forme au moyen d'un double-clic;
- protéger les formes pour empêcher la modification de leur format à l'aide de la souris (en faisant glisser les poignées de sélection pour les redimensionner);
- adapter la largeur d'une forme à celle de son bloc de texte;
- déterminer les points de connexion particuliers à chaque forme.

Le projet VBA, sous la forme de macros, de procédures, de fonctions et de morceaux de code, vérifie en partie les contraintes relatives aux classes spécifiques, ainsi que les propriétés des classes générales, des méta-classes et des classes spéciales dont elles héritent. Il contrôle également certains aspects qui touchent à la construction du modèle graphique de "parts". Enfin, il permet de traduire ce modèle en langage AlbertII.[23]

3.1.3. Les règles de réutilisation du sous-ensemble du pattern *Part*

Pour construire des modèles graphiques de "parts" à partir du méta-modèle sous-ensemble, nous empruntons les règles de réutilisation du pattern *Part* présentées au premier chapitre. Toutefois, nous n'employons que la deuxième méthode qui consiste à instancier les méta-classes et les méta-classes d'attributs pour définir les classes et les classes d'attributs spécifiques à l'application. Celles-ci héritent automatiquement de toutes les propriétés des classes générales et des classes générales d'attributs correspondantes.

La première méthode consistait à copier/coller les classes générales (d'attributs) et à s'en servir comme des classes régulières. Mais ce procédé comportait deux contraintes importantes : d'une part, la méta-classe (d'attributs) correspondante devait être copiée dans le modèle et, d'autre part, la classe générale (d'attributs) devait contenir comme instances toutes les occurrences de part, d'état (ou d'attribut) d'un type donné, faisant des éventuelles classes spécifiques (d'attributs) de ce type ses sous-classes. Ces contraintes devaient être traduites dans le projet VBA. En outre, nous ne percevions pas l'intérêt d'ajouter des classes supplémentaires alors que, par définition, les classes spécifiques (d'attributs) héritaient déjà des méta-classes (d'attributs) et étaient déjà les sous-classes des classes générales (d'attributs). Nous avons donc estimé que ce procédé n'enrichissait pas notre modélisation.

La troisième méthode, intitulée substitution de paramètre, consistait à remplacer une classe générale par une autre classe composée d'attributs ou de contraintes supplémentaires. Mais puisque les classes spécifiques héritent des propriétés des classes générales et que celles-ci trouvent leur correspondance dans le projet VBA de Visio, notre modèle de dessin rend la procédure de substitution impossible à réaliser. Elle contraindrait en effet l'utilisateur à modifier le code VBA. Cette faiblesse représente une véritable restriction tant au niveau de la construction du modèle qu'au niveau du sens que le concepteur souhaite lui conférer car elle empêche l'ajout de contraintes autres que celles prévues par le sous-ensemble du pattern *Part*. Cette restriction peut être contournée pour les classes d'attributs même si cela complique le dessin. En effet, si l'utilisateur souhaite, par exemple, ajouter une classe d'attributs à toutes les classes de parts, il peut créer une super classe (de parts mixtes) qui chapeaute l'ensemble des classes de parts et lui associer cette classe d'attributs. La nouvelle super classe racine se substitue en quelque sorte à la classe générale de parts.

Les classes spéciales de parts composées, de dispositifs géométriques et d'attributs qui ont été intégrées au méta-modèle sous-ensemble sont spécialisées pour définir des classes

[23] Voir la présentation du code du projet VBA du modèle "Systèmes manufacturiers" à l'annexe F.

spécifiques. Toutefois, au niveau du projet de dessin, les formes représentant les sous-ensembles des classes spéciales sont instanciées à partir des formes de base du gabarit "Gabarit_Pile_Conteneur_Tampon". Nous respectons ainsi l'idée émise dans la thèse selon laquelle les classes spéciales sont inclues dans le pattern *Part* pour être réutilisées par instanciation de la méta-classe d'attributs sous-classes (*subclass*). Mais dans notre configuration, les classes d'attributs sous-classes qui associent les classes spécifiques aux classes spéciales sont implicites.

3.1.4. Les contraintes du sous-ensemble et du modèle de dessin Visio

Les origines des contraintes qui s'exercent sur les modèles graphiques de "parts" sont au nombre de trois. Une partie de ces contraintes proviennent du pattern *Part* : la plupart ont été mentionnées de manière explicite dans le premier chapitre. D'autres sont particulières au sous-ensemble du pattern *Part* défini dans ce chapitre : certaines ont été explicitées dans les sections précédentes. Enfin, plusieurs contraintes sont liées plus spécifiquement aux caractéristiques de la modélisation graphique des "parts" dans Visio.

Toutes ces contraintes peuvent être réparties en trois catégories :
- celles qui sont vérifiables par le projet de dessin Visio;
- celles qui sont traduisibles en langage AlbertII;
- celles qui sont invérifiables et intraduisibles.

3.1.4.1 Les contraintes vérifiées au sein du projet

Ces contraintes sont vérifiées par les formules des feuilles ShapeSheet, par les choix liés à la configuration du modèle de dessin ou – principalement – par les macros du projet VBA.[24] Parmi celles qui sont contrôlées par le projet VBA, seules la contrainte n° 1 sur l'unicité du nom des classes et une partie de la contrainte n° 25 sur la syntaxe des classes d'attributs le sont au moyen d'un gestionnaire d'événements[25]. Toutes les autres sont testées lors de la traduction du modèle graphique de "parts" en langage AlbertII. Elles ne peuvent en effet être vérifiées que lorsque le modèle est censé être complet.

1. **Le caractère unique du nom des classes spécifiques**

Une classe est identifiée par son nom. Le caractère unique de ce nom est vérifié à trois moments par le projet VBA :
- à l'ouverture d'un projet de dessin contenant déjà un modèle graphique de "parts" si l'utilisateur le souhaite (un message lui est envoyé : voir la figure 3.11 dont l'exemple est emprunté à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D);
- lors de la création d'une classe : une première fois au moment où la forme est glissée ou collée sur la page de dessin puis une seconde fois lorsque la boîte de dialogue des propriétés personnalisées est fermée;

[24] Voir la présentation du code du projet VBA du modèle "Systèmes manufacturiers" à l'annexe F.
[25] Un événement est une occurrence ou une notification qui peut déclencher des réponses.

- après la modification du nom d'une classe existante au moyen de la "Fenêtre Propriétés Personnalisées".

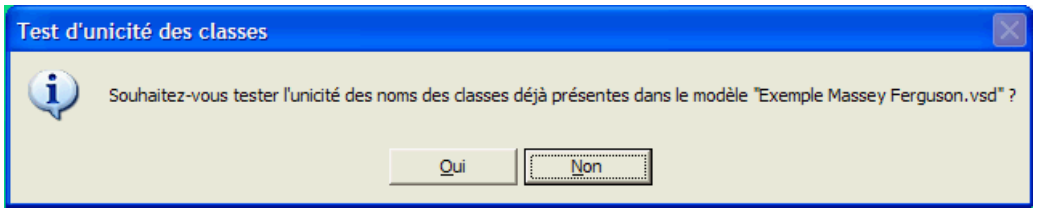


Figure 3.11 : Exemple de demande de vérification de l'unicité des noms des classes présentes au sein du modèle "Exemple Massey Ferguson" lors de son ouverture

La vérification est effectuée par la procédure "vérifierUnicitéNom" située dans le module de classe "ThisDocument". Cette procédure peut être déclenchée par l'un des trois événements liés à :

- l'ouverture d'un document : premier des trois moments cités ci-dessus;
- la création d'une forme : deuxième moment (la première fois lors de l'instanciation ou du collage);
- la modification de la formule de la cellule de la feuille ShapeSheet contenant le nom de la classe : deuxième moment (la seconde fois lors de la fermeture de la boîte de dialogue) et troisième moment.

La double vérification lors de la création d'une classe est importante parce que si la première vérification par ordre chronologique était supprimée et si l'utilisateur fermait la boîte de dialogue des propriétés personnalisées en cliquant sur "Annuler" au lieu de "OK", la classe recevrait un nom par défaut et la procédure de vérification ne serait pas exécutée.

Une exception à l'unicité du nom est tolérée pour les classes dont le nom est emprunté aux types de données élémentaires prédéfinis en langage AlbertII : il s'agit des classes d'identités, des classes de positions et des classes de dispositifs de valeur.

Puisqu'un modèle graphique de "parts" peut être construit sur plusieurs pages, nous laissons à l'utilisateur la faculté de reproduire plusieurs fois la même classe au sein du projet à condition de n'employer qu'une seule des formes représentant cette classe par page. Cette technique permet à l'utilisateur de découper le modèle graphique de "parts" en plusieurs sous-modèles qui peuvent être classés par thèmes comme le montre l'exemple relatif au tracteur Massey Ferguson à la section 3 (représentation graphique) de l'annexe D. La procédure "vérifierUnicitéNom" tient compte de cette possibilité : elle examine si les classes, dont le nom défini par l'utilisateur est identique, sont situées sur des pages différentes et si elles sont des instances de la même méta-classe ou des sous-ensembles de la même classe spéciale. Un message avertit l'utilisateur lorsqu'il ajoute au modèle graphique de "parts" une classe qui est déjà représentée par une forme. Voir la figure 3.12 dont l'exemple est emprunté à l'exemple de modélisation du tracteur Massey Ferguson.

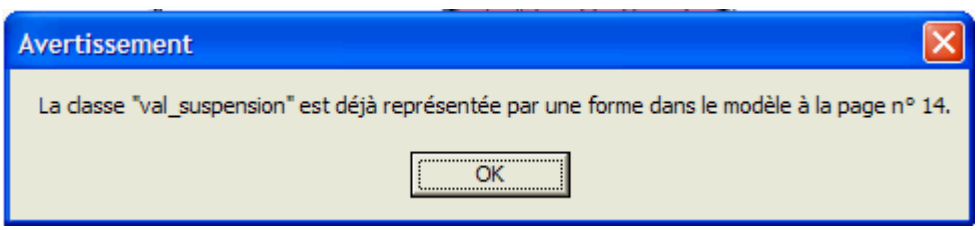


Figure 3.12 : Exemple de message envoyé à l'utilisateur chaque fois qu'il ajoute au modèle la classe d'attributs dispositifs de valeur *val_suspension* lorsqu'elle est déjà représentée

Lorsque le nom d'une classe ne vérifie pas la contrainte d'unicité, une boîte de dialogue en informe l'utilisateur et lui demande d'introduire une nouvelle dénomination. Voir la figure 3.13 dont l'exemple est emprunté à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D. Cette boîte reste visible à l'écran tant que l'utilisateur n'a pas encodé une chaîne de caractères non vide. Après la correction, la procédure de vérification est relancée au niveau du code car, bien que l'ancien nom ait été remplacé par le nouveau dans la cellule de la feuille ShapeSheet, le gestionnaire de l'événement lié à la modification de la formule d'une cellule ne se déclenche pas.

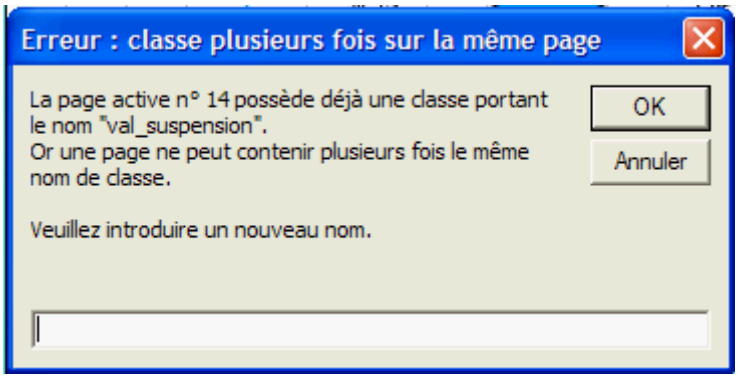


Figure 3.13 : Exemple de boîte de dialogue pour la correction de la contrainte d'unicité sur le nom de la classe d'attributs dispositifs de valeur *val_suspension*

2. **La syntaxe du nom des classes spécifiques**

La procédure "vérifierUnicitéNom" citée à la contrainte précédente vérifie aussi que l'utilisateur a introduit un nom de classe syntaxiquement correct : celui-ci ne peut pas être vide, ni commencer ou se terminer par un espace, ni, pour éviter toute ambiguïté au niveau des types de données AlbertII, être équivalent au mot-clé "*UNDEF*".

La procédure contrôle également que seules les classes d'identités, les classes de positions et les classes de dispositifs de valeur reçoivent pour nom un type prédéfini.

Si la contrainte n'est pas vérifiée, une boîte de dialogue informe l'utilisateur sur le type d'erreur détecté et lui demande d'introduire un nouveau nom. Voir en exemple la figure 3.14. L'intégralité de la procédure de vérification est ensuite relancée.

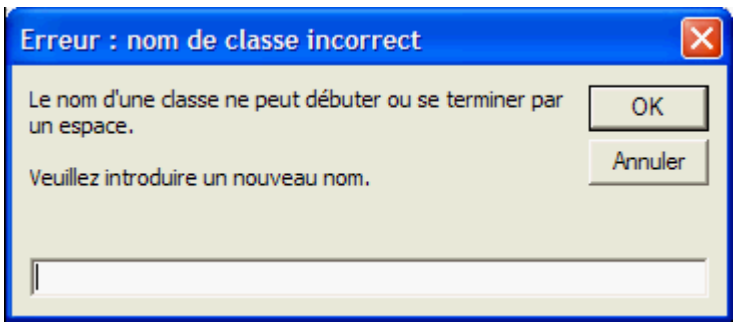


Figure 3.14 : Exemple de boîte de dialogue pour la correction
d'une erreur de syntaxe sur le nom d'une classe

3. **La définition de la part de base et de la part composée**

Une part de base est une part qui n'est pas constituée d'autres parts tandis qu'une part composée est un agrégat formé d'un ensemble d'autres parts.

Au moment de la traduction du modèle graphique de "parts" en langage AlbertII, un morceau de code du projet VBA vérifie qu'aucune classe de parts de base ne possède de classes d'attributs composants et que toutes les classes de parts composées au sens large sont au moins associées à une classe de parts composantes (ou contenues pour les classes de conteneurs).

4. **La partition de la classe générale de parts**

La première contrainte sur les classes générales stipule que les classes générales de parts de base, de parts composées et de tampons forment une partition de la classe générale des parts : une part ne peut pas être à la fois une part de base, une part composée et un tampon ou deux de ces types de parts mais elle doit être l'un des trois. Une partition est une super classe abstraite caractérisée par un ensemble de sous-classes exclusives.

La découpe en classes adoptée pour construire les modèles graphiques de "parts" attribue un caractère exclusif aux classes générales de parts de base, de parts composées et de tampons. En effet, les gabarits du modèle de dessin "Systèmes manufacturiers" imposent une forme différente pour créer (instancier ou spécialiser) une classe de parts de base, sous-classe de la classe générale de parts de base, une classe de parts composées au sens large, sous-classe de la classe générale de parts composées, ou une classe de tampons, sous-classe de la classe générale de tampons. En conséquence, une part ne peut pas être simultanément une part de base, une part composée et un tampon ou deux de ces trois types de parts. Rappelons que les classes de piles et de conteneurs sont des sous-classes des classes spéciales du même nom qui sont elles-mêmes des sous-classes de la classe générale de parts composées. La première partie de la contrainte est vérifiée.

Un sixième type de classes de parts existe : la classe de parts mixtes, sous-classe de la classe générale de parts mixtes. Mais elle se rencontre uniquement dans les hiérarchies de spécialisation. Or, comme nous l'avons indiqué dans la section consacrée aux méta-classes du sous-ensemble du pattern *Part*, les classes de parts mixtes ne peuvent être que des super classes abstraites et les sous-classes feuilles, qui sont des classes de parts de base, de parts composées ou de piles, déterminent le type des instances de ces super classes.[26] Un septième type de classes de parts pourrait être incarné par les instances de la méta-classe de parts. Mais celle-ci est une classe abstraite : elle est représentée par un calque dans le modèle de dessin et ne peut donc pas être instanciée. Seules les méta-classes de parts de base, de parts composées, de parts mixtes et de tampons peuvent être instanciées pour créer des classes spécifiques de parts qui sont des classes de parts de base, de parts composées au sens large, de parts mixtes ou de tampons. En conclusion, puisque les parts mixtes seules n'existent pas, une part doit être aussi au moins une part de base, une part composée, une pile, un conteneur ou un tampon. La seconde partie de la contrainte est vérifiée.

[26] La classe générale de parts de base et la classe générale de parts composées forment aussi une partition de la classe générale de parts mixtes. Les parts mixtes sont donc toujours aussi des parts de base, des parts composées ou des piles. Toutefois, dans le monde réel, les parts mixtes n'existent pas.

5. **Aucune part ni aucune pile n'est composée d'elle-même**

La deuxième contrainte relative aux classes générales et la cinquième contrainte relative aux classes spéciales sont vérifiées par le projet VBA lors de la traduction du modèle graphique de "parts" en langage AlbertII. La méthode consiste à éliminer les circuits[27] dans les relations composants. Cependant, cette vérification est rendue complexe par la définition de la spécialisation car une sous-classe hérite des classes d'attributs composants de sa super classe. Nous devons par conséquent interdire les circuits mixtes (relations sous-ensembles et composants).

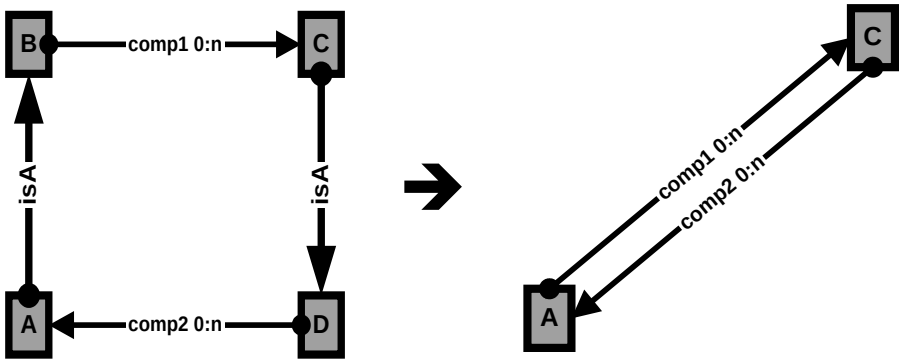


Figure 3.15 : Exemple de part composée d'elle-même dans une hiérarchie de spécialisation formant un circuit mixte (relations sous-ensembles et composants)

La figure 3.15 illustre par un exemple une hiérarchie de spécialisation dans laquelle une part est composée d'elle-même. Soit quatre classes de parts composées nommées *A*, *B*, *C* et *D*. *A* hérite de *B* qui est associé à *C* dans une relation de composition et *C* hérite de *D* qui est associé à *A* dans une relation de composition. Par conséquent, *A* hérite de *B* la classe d'attributs composants *comp1* qui la lie à *C* et *C* hérite de *D* la classe d'attributs composants *comp2* qui l'attache à *A*. Un circuit se forme donc au sein des relations de composition : chaque instance de la classe *A* (ou *C*) est composée d'elle-même. Dans ce cas, nous pouvons même parler de circuit mixte dans un graphe orienté. Par ailleurs, si le sens de la relation *comp2* avait été inversé, nous aurions obtenu un cycle[28] mixte qui se serait résumé à une double relation de composition ayant *A* comme origine et *C* ou *D* comme destination.[29]

[27] Un circuit dans un graphe orienté est un chemin simple qui possède le même sommet aux deux extrémités, c'est-à-dire que l'extrémité initiale du premier arc (flèche) est identique à l'extrémité finale du dernier. Un chemin est dit "simple" si chaque arc du chemin est emprunté une seule fois. [Sciences 2002]

[28] Le modèle graphique de "parts" doit être considéré ici comme un graphe non orienté. Un cycle est l'équivalent d'un circuit pour les graphes non orientés. La définition formelle est exactement la même dans les deux cas, sauf que l'on parle de chaîne et non de chemin et d'arête (qui a une direction mais pas de sens) au lieu d'arc. [Sciences 2002]

[29] En effet, comme les instances de *A* sont constituées d'instances de *C* et de *D* et comme les instances de *C* sont aussi des instances de *D* parce que *C* hérite de *D*, les classes d'attributs composants *comp1* et *comp2* sont redondantes. Ainsi que nous l'avons indiqué lors de la présentation des méta-classes du sous-ensemble, lorsque des instances d'une classe de parts composées sont constituées des instances d'une super classe de parts, elles peuvent être constituées de n'importe quelles instances des sous-classes de cette super classe. En outre, la configuration particulière du graphe de la figure 3.15 fait que toutes les instances de la super classe *D* sont également des instances de la sous-classe *C*. Nous pouvons donc parler dans ce cas de double relation de composition tant vers la sous-classe *C* que la super classe *D*.

Or, cette configuration est interdite par la cinquième contrainte relative aux méta-classes qui précise que deux instances de méta-classe de parts ou d'états ne peuvent être associées que par maximum une instance de (chaque) méta-classe d'attributs.

Une super classe ne peut être associée à aucune de ses sous-classes dans une relation de composition. En effet, comme la sous-classe hérite des classes d'attributs de ses super classes, elle va acquérir une classe d'attributs composants dont elle va devenir à la fois l'origine et la destination.

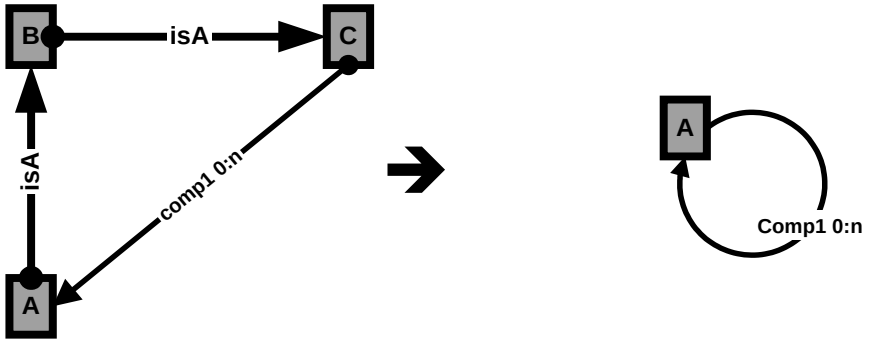


Figure 3.16 : Exemple de part composée d'elle-même dans une hiérarchie de spécialisation lorsqu'une sous-classe est un composant direct d'une de ses super classes

La figure 3.16 illustre ce cas par un exemple dont les classes sont reprises de l'exemple de la figure 3.15. Le schéma présenté ici est relativement simple. Il peut être complexifié en ajoutant un nombre illimité de classes de parts composées entre la super classe *C* et la sous-classe *A*, toutes associées au moyen de classes d'attributs composants et sous-classes. La figure 3.15 présente un exemple de ce type si l'on se place du point de vue de la super classe *B* et de la sous-classe *A* ou de la super classe *D* et de la sous-classe *C*. La vérification de cette contrainte consiste ici aussi à interdire les circuits mixtes.

Une sous-classe ne peut pas non plus être associée à une de ses super classes dans une relation de composition. En effet, comme les instances d'une classe de parts composées (ou de piles) qui est associée à une super classe de parts composantes peuvent être constituées d'instances de n'importe quelles sous-classes de cette super classe, la sous-classe de parts composées (ou de piles) peut être composée d'elle-même. En pratique, cela revient à proscrire les cycles mixtes (relations sous-ensembles et composants) dans les modèles graphiques de "parts". Remarquons à ce sujet que le graphe des classes générales du sous-ensemble du pattern *Part* (figure 3.1) contient un cycle mixte formé par les classes générales de parts et de parts composées. Mais ce schéma ne constitue qu'une représentation simplifiée de la réalité. Il est d'ailleurs accompagné d'une liste de contraintes non formelles.

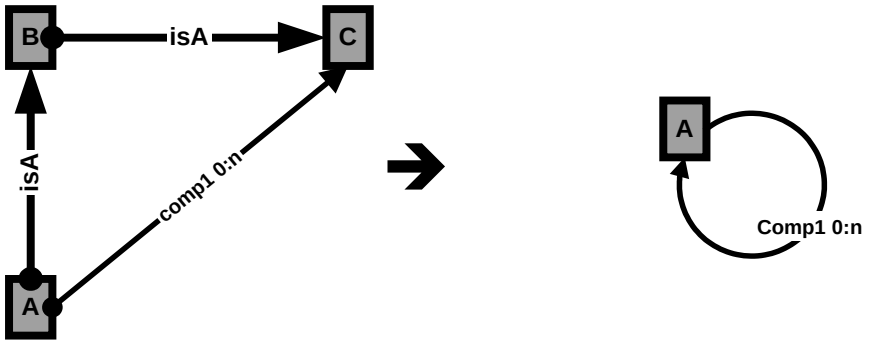


Figure 3.17 : Exemple de part composée d'elle-même dans une hiérarchie de spécialisation lorsqu'une super classe est un composant d'une de ses sous-classes

La figure 3.17 illustre ce cas par un exemple dont les classes sont reprises de l'exemple de la figure 3.15. Comme pour le schéma précédant, un nombre illimité de classes de parts composées peuvent être ajoutées entre la super classe *C* et la sous-classe *A* en les unissant au moyen de classes d'attributs composants et sous-classes à condition que *C* soit la destination d'une classe d'attributs composants. Nous insistons sur le fait que le graphe de droite de la figure 3.17 peut être déduit du graphe de gauche en raison de la configuration particulière de ce dernier dans laquelle toutes les instances de *C* sont des instances de *A* et inversement. En effet, si la super classe de parts composantes possédait des sous-classes situées hiérarchiquement au même niveau – par exemple une classe de parts composées *D*, sous-classe directe de *C* ou de *B* –, les instances de la sous-classe de parts composées pourraient être constituées d'instances d'autres sous-classes plutôt que de sa propre sous-classe – dans notre exemple, une instance de la sous-classe *A* pourrait être constituée d'une instance de la sous-classe *D*. Afin d'éviter toute ambiguïté, ainsi que l'obligation d'exprimer cet interdit (une part ou une pile ne peut être composée d'elle-même) en langage AlbertII, nous prohibons tous les cycles mixtes dans les modèles graphiques de "parts".[30]

6. **La partition de la classe générale de dispositifs**

Selon la troisième contrainte relative aux classes générales, un dispositif ne peut être simultanément un dispositif physique, un dispositif géométrique et un dispositif de fixation ou deux de ces trois types de dispositifs mais il doit être l'un des trois.

La première partie de cette contrainte est vérifiée par la configuration de la modélisation graphique des "parts" parce que les gabarits du modèle de dessin "Systèmes manufacturiers" imposent une forme différente pour créer (instancier) une classe de dispositifs physiques, géométriques (qui peut être une classe de dispositifs de contiguïté ou d'enclos) ou de fixation qui constitue automatiquement une sous-classe respectivement de la classe générale de dispositifs physiques, géométriques ou de fixation.

La second partie est vérifiée aussi parce que la classe générale de dispositifs constitue une classe abstraite : la méta-classe de dispositifs ne peut être instanciée et toutes les instances des méta-classes de dispositifs physiques, géométriques et de fixation sont des sous-classes des classes générales de dispositifs physiques, géométriques et de fixation. En conclusion, un dispositif doit être soit un dispositif physique, soit un dispositif géométrique, soit un dispositif de fixation.

À l'image des modèles graphiques de "parts", la sémantique des classes générales du sous-ensemble est définie comme une partition dans les hiérarchies de spécialisation. Ce

[30] En réalité, il ne s'agit pas de proscrire tous les cycles mixtes mais plutôt d'interdire les graphes qui offrent la possibilité de partir d'une classe de parts composées ou de piles et d'aboutir à une autre classe de parts en empruntant des chemins différents et en suivant le sens des relations composants et sous-ensembles : les graphes doivent comprendre au moins un chemin composé uniquement de relations sous-ensembles et un chemin qui peut être mixte mais qui doit se terminer par un arc représentant une relation composant. Toutefois, pour une question de facilité, nous utiliserons l'expression "cycle mixte" pour désigner les classes de parts composées d'elles-mêmes en dehors des circuits (mixtes) présentés ci-dessus.

principe est aussi valable pour les (fausses) classes générales d'états, d'attributs, de sous-classes et de commentaires.[31]

7. **Chaque part possède une identité (distincte)**

Le modèle de dessin vérifie que chaque part possède une identité, tandis que le caractère unique de cette identité fait l'objet d'une contrainte rédigée en langage AlbertII. La contrainte au niveau du modèle de dessin subit un double contrôle : d'abord de la part du projet VBA qui oblige chaque classe de parts à être associée au minimum à une classe d'identités, soit de manière directe, soit par l'intermédiaire du mécanisme d'héritage dans le cas des sous-classes. Un second contrôle est exercé par la configuration même de la modélisation graphique des "parts" dans Visio qui fixe la cardinalité des classes d'attributs identités à "1:1", de sorte que chaque instance d'une classe de parts soit associée à une et une seule instance de chacune des classes d'identités auxquelles sa classe est liée. Ce double contrôle permet de vérifier la première partie de la sixième contrainte sur les classes générales.

8. **La partition de la méta-classe de parts**

Selon la première contrainte relative aux méta-classes, une classe de parts ne peut être simultanément une classe de parts mixtes, une classe de parts de base, une classe de parts composées et une classe de tampons ou deux ou trois de ces quatre classes mais elle doit être l'une des quatre.

Le caractère exclusif de ces quatre méta-classes est vérifié par la configuration de la modélisation graphique des "parts" parce que, comme nous l'avons mentionné à la contrainte n° 4 consacrée à la partition de la classe générale de parts, une classe de parts ne peut être instanciée qu'à partir d'une des quatre formes des gabarits du modèle de dessin "Systèmes manufacturiers" correspondant chacune à l'un des quatre types de méta-classes de parts. Quant aux deux classes spéciales de piles et de conteneurs représentées par deux formes du gabarit "Gabarit_Pile_Conteneur_Tampon", elles sont toutes, avec leurs sous-classes, des instances de la méta-classe de parts composées.

Les méta-classes de parts de base, de parts composées, de parts mixtes et de tampons sont non seulement des méta-classes exclusives mais elles forment aussi une partition de la méta-classe de parts. En effet, comme nous l'avons démontré à la contrainte n° 4, celle-ci constitue une classe abstraite : elle est représentée par un calque dans le modèle de dessin et ne peut donc pas être instanciée pour créer des classes spécifiques de parts.

9. **La partition de la méta-classe de dispositifs**

La deuxième contrainte sur les méta-classes exige que les méta-classes de dispositifs physiques, géométriques et de fixation forment une partition de la méta-classe de dispositifs.

La configuration de la modélisation graphique des "parts" permet de vérifier implicitement le caractère exclusif des trois types de méta-classes de dispositifs parce qu'une classe de dispositifs ne peut être instanciée qu'à partir d'une des trois formes du gabarit "Gabarit_SysMan" du modèle de dessin "Systèmes manufacturiers" correspondant chacune à l'un des trois types de méta-classes de dispositifs. Quant aux deux classes spéciales de dispositifs de contiguïté et de dispositifs d'enclos représentées par deux formes du gabarit "Gabarit_Pile_Conteneur_Tampon", elles sont toutes, avec leurs sous-classes, des instances

[31] Chaque méta-classe possède une classe correspondante parmi les classes générales du sous-ensemble. Les (fausses) méta-classes d'états, d'attributs, de sous-classes et de commentaires incarnées par les calques induisent donc l'existence de (fausses) classes générales d'états, d'attributs, de sous-classes et de commentaires.

de la méta-classe de dispositifs géométriques. Ces trois méta-classes forment également une partition de la méta-classe de dispositifs car celle-ci est une classe abstraite à l'image de la méta-classe de parts. À l'instar des modèles graphiques de "parts" et des classes générales du sous-ensemble, la sémantique des méta-classes est définie comme une partition dans les hiérarchies de spécialisation. Ce principe est aussi valable pour les (fausses) méta-classes d'états, d'attributs, de sous-classes et de commentaires.

10. **Une classe de parts composées commune pour les classes de parts composantes associées à une classe de dispositifs géométriques**

Selon la troisième contrainte relative aux méta-classes, toutes les classes de parts composantes associées à une classe de dispositifs géométriques doivent être associées au travers de relations composants à la classe de parts composées qui est définie par cette classe de dispositifs géométriques.

Le projet VBA vérifie cette contrainte au moment de la traduction du modèle graphique de "parts" en langage AlbertII. La figure 3.18 montre le type de message envoyé à l'utilisateur lorsque cette contrainte n'est pas vérifiée. L'exemple est emprunté à celui de la modélisation du tracteur Massey Ferguson en présupposant que le moteur du plus petit des trois modèles de tracteurs assemble de gros pistons au lieu de petits.

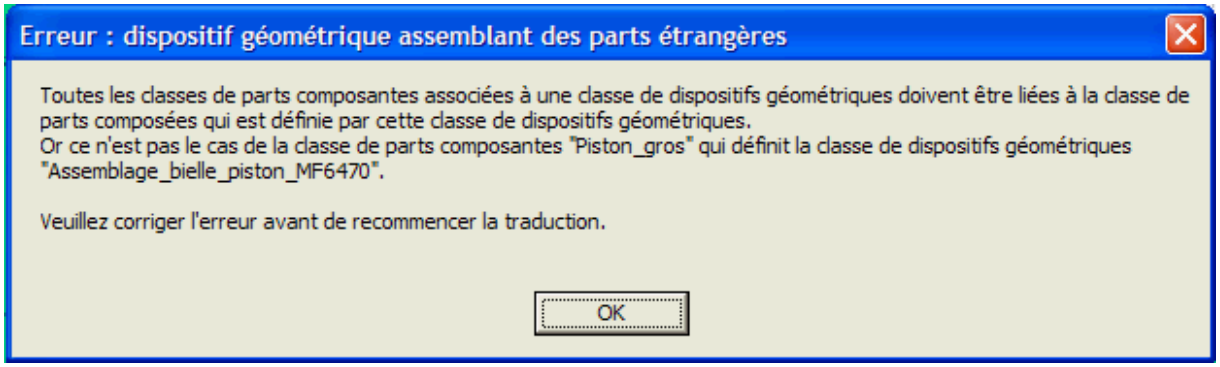


Figure 3.18 : Exemple de message envoyé lorsque la troisième contrainte relative aux méta-classes (dispositifs géométriques) n'est pas vérifiée (contrainte n° 10)

Le principe de l'héritage dans les hiérarchies de spécialisation complique cependant cette vérification. La classe de parts composées commune doit donc être associée à toutes les classes de parts composantes soit directement, soit par l'intermédiaire de ses super classes dont elle hérite des classes d'attributs composants. Elle-même et ses super classes éventuelles peuvent également être associées aux super classes des classes de parts composantes unies à la classe de dispositifs géométriques. La classe de parts composées commune peut également être une super classe qui ne possède pas de classes d'attributs composants mais dont chaque sous-classe feuille possède des classes d'attributs composants dont la destination sont les classes de parts composantes qui définissent la classe de dispositifs géométriques associée à cette super classe de parts composées. Une seule classe peut répondre à ce critère car une classe de dispositifs géométriques ne peut être jointe qu'à une classe de parts composées unique.

Le projet VBA vérifie également que les bornes de la cardinalité de la classe d'attributs dispositifs géométriques ne sont pas plus élevées que les bornes de la cardinalité de la classe

d'attributs composants dont la destination est la même que celle de la classe d'attributs dispositifs géométriques (en tenant compte de l'héritage).

11. **Une classe de parts composées commune pour les classes de parts composantes associées aux classes de dispositifs géométriques d'une classe de dispositifs de fixation**

La quatrième contrainte sur les méta-classes requiert que toutes les classes de parts composantes associées aux classes de dispositifs géométriques d'une classe de dispositifs de fixation soient associées au travers de relations composants à la classe de parts composées qui est définie par ces classes de dispositifs géométriques.

La contrainte précédente étant préalablement contrôlée lors de la traduction du modèle graphique de "parts" en langage AlbertII, le projet VBA vérifie simplement que toutes les classes de dispositifs géométriques qui définissent une classe de dispositifs de fixation définissent aussi la même classe de parts composées. Ici aussi une seule classe de parts composées commune est possible. Le projet VBA vérifie également que les bornes de la cardinalité de la classe d'attributs dispositifs de fixation ne sont pas plus élevées que les bornes de la cardinalité de la classe d'attributs géométries dont la destination est la même que celle de la classe d'attributs dispositifs de fixation.

12. **Une seule classe d'attributs associe deux classes de parts ou d'états**

La cinquième contrainte affectée aux méta-classes précise que deux instances de méta-classes de parts ou d'états ne peuvent être associées que par maximum une instance de chaque méta-classe d'attributs. Nous allons plus loin en affirmant qu'il ne peut exister qu'une seule classe d'attributs entre deux classes de parts ou d'états. D'abord parce que la contrainte évoquée ci-dessus exige qu'une seule instance d'une même méta-classe d'attributs attache deux instances de méta-classe, quel que soit le sens de la relation, et que cette contrainte est vérifiée par le projet VBA lors de la traduction du modèle graphique de "parts" en langage AlbertII. Ensuite parce que chaque méta-classe d'attributs associe deux méta-classes de parts ou d'états différentes et que ce principe se répercute sur les instances de ces méta-classes. Ainsi, une classe d'attributs identités réunit une classe de parts et une classe d'identités, tandis qu'une classe d'attributs valeurs relie une classe de dispositifs physiques à une classe de dispositifs de valeur, etc.

Les classes d'attributs composants, contenus, contenants et sous-classes constituent les seules exceptions car elles unissent toutes les quatre des classes de parts au sens général du terme. Deux doubles configurations sont alors possibles : la classe d'attributs composants/contenus/contenants et la classe d'attributs sous-classes possèdent la même origine et la même destination ou l'origine de l'une est la destination de l'autre et réciproquement. Or, ces configurations sont interdites car les classes d'attributs formeraient un circuit ou un cycle mixte qui enfreindrait la deuxième contrainte relative aux classes générales (une part ne peut être composée d'elle-même), la cinquième contrainte relative aux classes spéciales (une pile ne peut être composée d'elle-même), la sixième contrainte relative aux classes spéciales (un tampon ne peut se contenir lui-même) ou une des propriétés du conteneur (un conteneur ne peut être ni un composant, ni un contenu). Voir à ce sujet la contrainte n° 5 ou la contrainte n° 14.

Le projet VBA vérifie donc qu'une seule classe d'attributs subsiste entre deux mêmes classes de parts ou d'états. D'autre part, puisque l'utilisateur peut représenter plusieurs fois la même classe sur le dessin, il contrôle également que toutes les classes d'attributs portant le même nom associent toujours les mêmes classes de parts et d'états et possèdent les mêmes valeurs de cardinalité, de duplication et d'ordre.

Le principe d'héritage des classes d'attributs au sein d'une hiérarchie de spécialisation pose ici aussi un problème particulier car il permet à une sous-classe de parts d'entretenir indirectement plusieurs relations avec la même classe. Trois configurations de ce type sont possibles au travers des relations composants, contenus ou contenants :

- lorsqu'une sous-classe de parts et une de ses super classes sont directement associées à une même classe de parts ou d'états;
- lorsqu'une classe de parts ou d'états est directement associée à une super classe de parts et à une sous-classe de celle-ci;
- lorsqu'une sous-classe de parts et une de ses super classes sont directement associées à une super classe de parts et à une sous-classe de celle-ci.

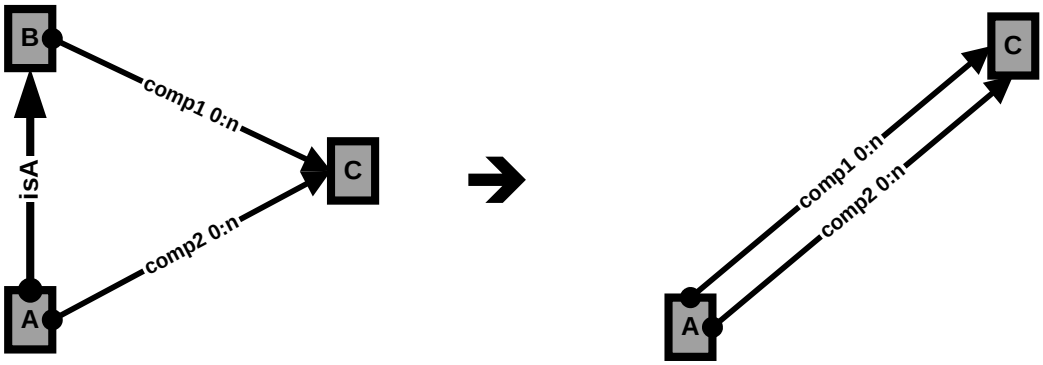


Figure 3.19 : Exemple de relations multiples entre une part composée membre d'une hiérarchie de spécialisation et une autre part composante

La figure 3.19 illustre la première configuration par un exemple. Soit trois classes de parts composées nommées *A, B,* et *C*. *A* hérite de *B* et toutes les deux sont associées à *C* dans une relation de composition. Par conséquent, *A* hérite de *B* la classe d'attributs composants *comp1* qui la lie à *C,* ce qui engendre une double relation ayant *A* comme origine et *C* comme destination. Nous aurions pu prendre des classes d'attributs contenus ou contenants à la place des classes d'attributs composants dans cet exemple mais pas un mélange des trois car cela aurait contrevenu à la définition de la spécialisation des classes de parts (voir à ce sujet le tableau 3.1).

Ces trois types de classes d'attributs ne sont pas les seuls en cause dans cette configuration particulière car le phénomène d'héritage permet également à une sous-classe de parts de posséder indirectement plusieurs classes d'attributs dispositifs physiques ayant la même classe de dispositifs physiques comme destination. Par contre, une sous-classe de piles ou de conteneurs ne peut être associée plus d'une fois à la même classe de dispositifs de contiguïté ou d'enclos car ces dernières ne peuvent être la destination que d'une seule classe d'attributs contiguïtés ou enclos. Une sous-classe de parts composées ne peut pas non plus être associée plus d'une fois à la même classe de dispositifs géométriques pour la même raison.

La figure 3.20 illustre la deuxième configuration par un exemple emprunté cette fois aux relations contenus. Soit trois classes de tampons nommées *A*, *B*, et *C*. *A* contient *B* et *C* qui est un sous-ensemble de *B*. La classe d'attributs contenus *cont1* exprime que les instances de *A* contiennent les instances de *B* et de sa sous-classe *C*. Par conséquent, elle fait double emploi avec la classe d'attributs contenus *cont2*, ce que traduit le graphe de droite.

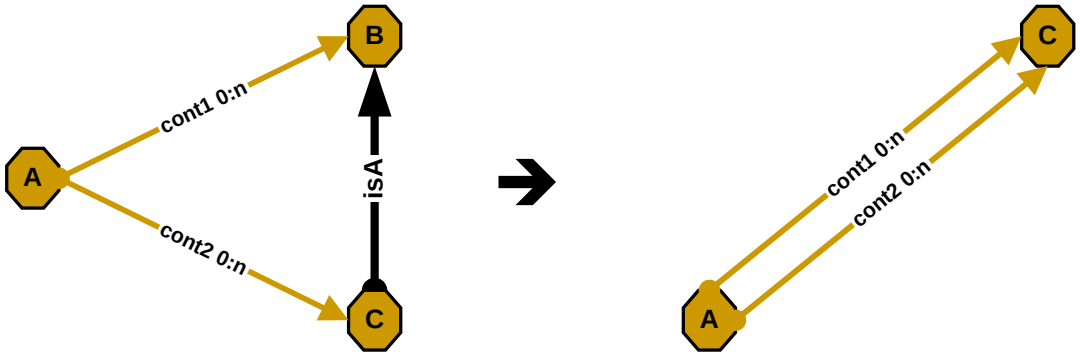


Figure 3.20 : Exemple de relations multiples entre un tampon et un autre tampon membre d'une hiérarchie de spécialisation

Une telle configuration peut également se produire au niveau des classes de dispositifs de contiguïté et d'enclos lorsqu'une de ces classes est directement associée à une super classe de parts composantes (pour les piles) ou contenues (pour les conteneurs) et à une sous-classe de celle-ci. Elle se présente aussi lorsqu'une classe de dispositifs géométriques est directement associée à une super classe de parts composantes et à une sous-classe de celle-ci.

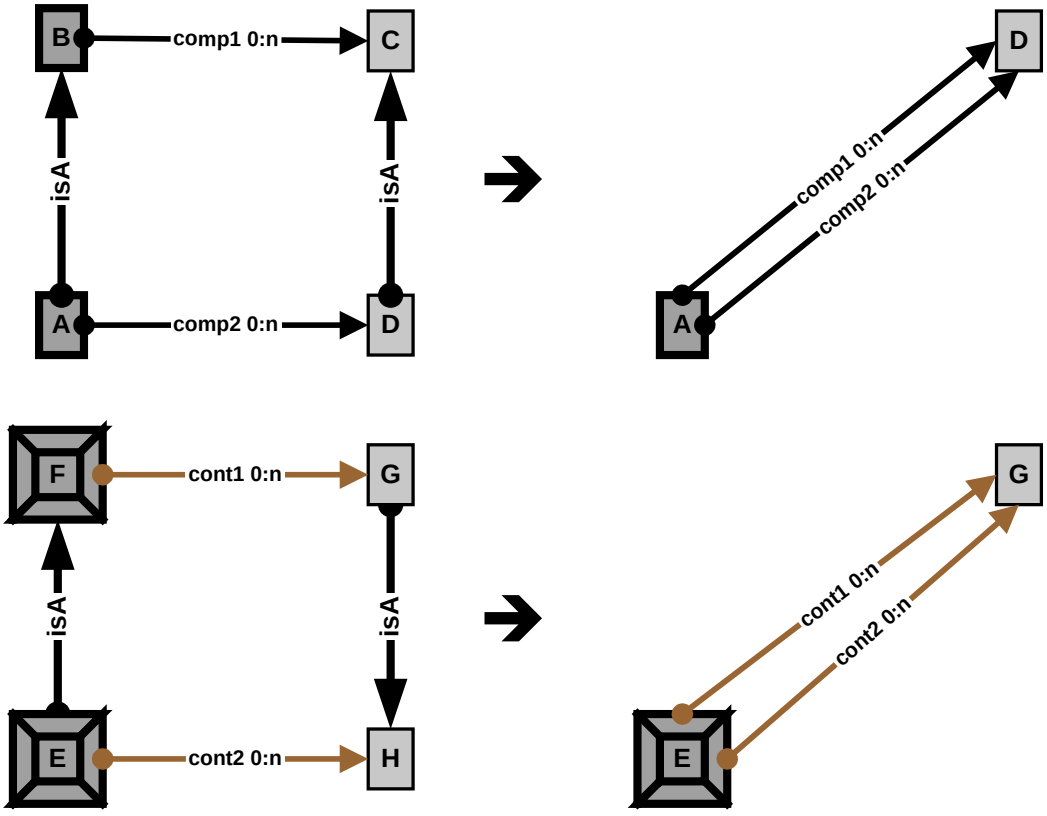


Figure 3.21 : Exemples de relations multiples entre une part composée ou un conteneur et une part de base, tous membres d'une hiérarchie de spécialisation

La figure 3.21 illustre la troisième configuration de deux manières différentes. La première est exprimée par les deux graphes situés dans la partie supérieure de la figure. La classe de parts composées *A* est constituée de la classe de parts de base *D* et est un sous-ensemble de la classe de parts composées *B* qui est elle-même constituée de la classe de parts de base *C* qui est la super classe de *D*. Par conséquent, *A* hérite de *B* la classe d'attributs composants *comp1* et nous retrouvons la même configuration que dans l'exemple précédent avec une double relation entre *A* et *D*. La seconde manière est exprimée par les deux graphes situés dans la partie inférieure de la figure. La classe de conteneurs *E* contient la classe de parts de base *H* et est un sous-ensemble de la classe de conteneurs *F* qui contient la classe de parts de base *G* qui est, cette fois, la sous-classe de *H*. Par conséquent, *E* hérite de *F* la classe d'attributs contenants *cont1* et nous nous retrouvons dans la même configuration que l'exemple précédent avec une double relation entre *E* et *G*.

À l'image des contraintes n° 5 et n° 14 qui utilisent l'expression "cycle mixte" pour indiquer qu'une part est composée ou contenue par elle-même, nous employons la même expression pour signifier l'existence de relations multiples soit entre une classe de parts et une autre classe de parts ou une classe de dispositifs physiques, soit entre une classe de dispositifs de contiguïté ou d'enclos et une classe de parts composantes ou contenues, soit entre une classe de dispositifs géométriques et une classe de parts composantes. Nous interdisons donc les cycles mixtes formés par, d'une part, des relations sous-ensembles et, d'autre part, des relations composants, contenus, contenants, dispositifs physiques, dispositifs géométriques, dispositifs de contiguïté ou dispositifs d'enclos exclusives.

13. **La hiérarchisation des sous-classes de parts**

La sixième contrainte relative aux méta-classes spécifie que les instances de la méta-classe d'attributs sous-classes forment un ordre partiel sur les classes de parts.

Le projet VBA vérifie l'absence de circuits dans le modèle graphique de "parts" lors de sa traduction en langage AlbertII. Nous pouvons en effet considérer dans ce cas-ci le modèle comme un graphe orienté. Toutefois, dans la pratique, le projet VBA utilise la même procédure que celle employée pour détecter les circuits et les cycles mixtes (relations composants et sous-ensembles) explicités dans la contrainte n° 5.

14. **Aucun tampon ne se contient lui-même.**

La sixième contrainte relative aux classes spéciales est vérifiée par le projet VBA lors de la traduction du modèle graphique de "parts" en langage AlbertII. Les configurations et les exemples présentés à la contrainte n° 5 (aucune part ni aucune pile n'est composée d'elle-même) et qui concernent les classes d'attributs composants peuvent être transposés aux classes d'attributs contenus dont l'origine sont des classes de tampons mais pas aux classes d'attributs contenants dont l'origine sont des classes de conteneurs car un conteneur ne peut jamais être un contenu puisqu'il représente une entité virtuelle :

- une classe de tampons ne peut faire partie d'un circuit composé de relations contenus;
- une classe de tampons ne peut faire partie d'un circuit composé de relations contenus et sous-ensembles;

- une super classe de tampons ne peut être associée à aucune de ses sous-classes dans une relation de contenance;
- une sous-classe de tampons ne peut être associée à aucune de ses super classes dans une relation de contenance.

La méthode consiste donc à éliminer les circuits uniformes dans les relations contenus, ainsi que les circuits et les cycles mixtes dans les relations contenus et sous-ensembles.

Nous ne devons pas nous préoccuper du mélange des relations de composition et de contenance car :

- un tampon ne peut être un composant et ne peut être le contenu que d'un autre tampon;
- un conteneur ne peut être ni un composant, ni un contenu;
- tandis qu'une part de base ou une part mixte ne peut contenir ou être composée d'autres parts.

Par contre, une classe de parts composées peut très bien être composée d'une classe de piles constituée de cette classe de parts composées. Mais les classes de parts composées et de piles ne présentent pas de danger à cet égard car elles ne possèdent que des relations de composition. Nous ne devons pas non plus tenir compte du mélange des relations sous-ensembles avec des relations de composition ou de contenance car les super classes et les sous-classes de parts doivent toutes être du même type, à l'exception des classes de parts mixtes mais celles-ci ne possèdent ni classes d'attributs composants ni classes d'attributs contenus.

15. Le caractère exclusif des deux types de classes spéciales de piles

Selon la huitième contrainte sur les classes spéciales, une pile ne peut être simultanément une pile FIFO et une pile LIFO.

Cette contrainte est vérifiée par la configuration de la modélisation graphique des "parts" parce qu'une propriété personnalisée de la forme représentant la classe spéciale de piles impose à l'utilisateur de choisir un type unique pour les instances de cette classe. En réalité, cette forme permet de représenter quatre types de classes de piles qui forment en quelque sorte une partition de la classe spéciale de piles : FIFO, LIFO ou standard (piles seulement ouvertes aux deux extrémités ou entièrement ouvertes selon la présence ou l'absence d'une classe de dispositifs d'enclos attachée à la classe de piles).

D'autre part, les classes de piles qui font partie de la même hiérarchie de spécialisation doivent toutes être du même type : FIFO, LIFO ou standard. En outre, les classes de piles qui ne possèdent pas de classe de dispositifs d'enclos ne peuvent être de type FIFO ou LIFO.

16. Le caractère exclusif des deux sous-classes spéciales de la classe générale de dispositifs géométriques

La neuvième contrainte relative aux classes spéciales exige qu'un dispositif géométrique ne puisse appartenir simultanément à la classe spéciale de dispositifs de contiguïté et à la classe spéciale de dispositifs d'enclos.

Cette contrainte est vérifiée par la configuration de la modélisation graphique des "parts" parce que le gabarit "Gabarit_Pile_Conteneur_Tampon" du modèle de dessin "Systèmes manufacturiers" impose une forme différente pour créer une classe de dispositifs de contiguïté ou d'enclos, instance de la méta-classe de dispositifs géométriques, qui constitue automatiquement une sous-classe respectivement de la classe spéciale de dispositifs de contiguïté ou d'enclos.

En réalité, au niveau du modèle de dessin "Systèmes manufacturiers"du logiciel Visio, une classe de dispositifs ne peut être instanciée qu'à partir d'une des cinq formes des deux gabarits correspondant chacune à l'une des cinq méta-classes ou classes spéciales de dispositifs. Les formes représentant les méta-classes de dispositifs physiques, géométriques et de fixation, ainsi que les classes spéciales de contiguïté et d'enclos, forment donc ici une partition de la méta-classe de dispositifs symbolisée par un calque auquel appartiennent exclusivement ces cinq formes.

17. **Le caractère exclusif des deux sous-classes spéciales de la classe générale d'attributs dispositifs géométriques**

Selon la dixième contrainte sur les classes spéciales, un attribut dispositif géométrique ne peut être simultanément un attribut dispositif de contiguïté et un attribut dispositif d'enclos.

Cette contrainte est vérifiée par la configuration de la modélisation graphique des "parts" parce que le gabarit "Gabarit_Pile_Conteneur_Tampon" du modèle de dessin "Systèmes manufacturiers" impose une forme différente pour créer une classe d'attributs dispositifs de contiguïté ou d'enclos, instance de la méta-classe d'attributs dispositifs géométriques, qui constitue automatiquement une sous-classe respective de la classe spéciale d'attributs dispositifs de contiguïté ou d'enclos.

18. **Le caractère exclusif des deux sous-classes spéciales de la classe générale d'attributs géométries**

Selon la onzième contrainte sur les classes spéciales, un attribut géométrie ne peut être à la fois un attribut contiguïté et un attribut enclos.

Cette contrainte est vérifiée par la configuration de la modélisation graphique des "parts" parce que le gabarit "Gabarit_Pile_Conteneur_Tampon" du modèle de dessin "Systèmes manufacturiers" impose une forme différente pour créer une classe d'attributs contiguïtés ou enclos, instance de la méta-classe d'attributs géométries, qui constitue automatiquement une sous-classe respective de la classe spéciale d'attributs contiguïtés ou enclos.

19. **Une classe de piles ou de conteneurs commune pour les classes de parts composantes ou contenues associées à une classe de dispositifs de contiguïté**

La douzième contrainte relative aux classes spéciales indique que chaque classe de parts qui est le composant d'une classe de piles doit être associée à au moins une classe de dispositifs de contiguïté de cette classe de piles, tandis qu'au moins une classe de parts contenue par une classe de conteneurs doit être associée à au moins une classe de dispositifs de contiguïté de cette classe de conteneurs. À l'inverse, chaque classe de dispositifs de contiguïté d'une classe de piles ne peut être associée qu'avec une ou plusieurs classes de parts qui sont les composants de cette classe de piles, tandis que chaque classe de dispositifs de contiguïté d'une classe de conteneurs ne peut être associée qu'avec une ou plusieurs classes de parts contenues par cette classe de conteneurs. Nous devons être attentifs au fait que les classes de dispositifs de contiguïté peuvent être héritées des super classes de piles ou de conteneurs et que les classes de parts composantes ou contenues qui définissent les classes de dispositifs de contiguïté d'une classe de piles ou de conteneurs peuvent être les sous-classes des classes de parts qui sont les composants ou les contenus de cette classe de piles ou de conteneurs.

Les parts qui constituent les composants d'une pile sont contiguës les unes aux autres. Nous pouvons donc faire un parallèle entre les attributs composants associant une pile à ses parts composantes et les attributs contiguïtés associant cette pile à ses dispositifs de contiguïté. Sachant que la cardinalité de toutes les classes d'attributs dispositifs de contiguïté est fixée à "2:2" pour exprimer que les parts sont contiguës deux à deux, la somme des bornes supérieures/inférieures des cardinalités des attributs composants cités précédemment doit être supérieure d'une unité à la somme des bornes supérieures/inférieures des cardinalités des attributs contiguïtés cités précédemment. Toutefois, ce principe ne peut être transposé au niveau des classes car la modélisation peut exiger que les cardinalités des classes d'attributs contiguïtés soient supérieures à celles des classes d'attributs composants. Prenons l'exemple d'une pile constituée de deux boulons et de deux vis. Si les deux boulons sont contigus et les deux vis aussi, la pile doit être associée à un dispositif de contiguïté associant les deux boulons, à un dispositif de contiguïté associant les deux vis et à un dispositif de contiguïté associant un boulon et une vis. Par contre, si les boulons et les vis sont placés en alternance, la pile doit être associée à trois dispositifs de contiguïté associant chacun un boulon et une vis. Par conséquent, si le nombre maximum de composants de la classe de piles est de quatre, la somme des bornes supérieures des cardinalités des classes d'attributs contiguïtés doit être égale à cinq afin de prendre en considération toutes les configurations possibles. En conclusion, la somme des bornes supérieures/inférieures des cardinalités des classes d'attributs contiguïtés citées précédemment ne peut être inférieure de plus d'une unité par rapport à la somme des bornes supérieures/inférieures des cardinalités des classes d'attributs composants citées précédemment.[32]

Mais cela ne suffit pas à garantir l'exactitude du parallélisme car si une classe de piles possède plusieurs classes d'attributs composants dont la borne supérieure de la cardinalité est un nombre indéfini ("N" ou "n"), la contrainte est vérifiée par l'algorithme si cette classe de piles possède au moins une classe d'attributs contiguïtés dont la borne supérieure de la cardinalité est un nombre indéfini ("N" ou "n"). Au niveau des classes de conteneurs, le même parallélisme peut être fait entre les classes d'attributs contenants et les classes d'attributs contiguïtés mais celui-ci n'est pas probant. En effet, comme les classes de parts contenues ne doivent pas toutes être associées à une classe de dispositifs de contiguïté, le nombre de dispositifs de contiguïté peut être inférieur de plus d'une unité par rapport au nombre de parts contenues. D'autre part, si toutes les parts d'un conteneur sont contiguës entre elles, le nombre de dispositifs de contiguïté vaut : (n - 1) ! avec n le nombre de parts contenues.

Lors de la traduction du modèle graphique de "parts" en langage AlbertII, le projet VBA vérifie pour chaque classe de piles que chacune de ses classes de parts composantes soit associée – même par héritage – à une de ses classes de dispositifs de contiguïté, que ses classes de dispositifs de contiguïté ne soient associées – même par héritage – qu'à ses classes de parts composantes, et que les cardinalités de ses classes d'attributs contiguïtés respectent ce qui a été énoncé précédemment. Le projet VBA vérifie également pour chaque classe de conteneurs que ses classes de dispositifs de contiguïté ne soient associées – même par héritage – qu'à ses classes de parts contenues. L'algorithme permet de vérifier des configurations complexes comme, par exemple, une classe de dispositifs de contiguïté d'une classe de piles qui serait définie par une classe de parts dont la super classe serait le composant d'une super classe de cette classe de piles.

[32] Si au moins une borne supérieure est un nombre indéfini ("N" ou "n"), la somme des bornes supérieures est aussi un nombre indéfini.

20. **Une classe de dispositifs d'enclos unique pour les classes de parts qui composent une classe de piles ou qui sont contenues par une classe de conteneurs**

La treizième contrainte relative aux classes spéciales indique que l'éventuelle classe de dispositifs d'enclos d'une classe de piles ou de conteneurs ne peut être associée qu'aux classes de parts qui composent cette classe de piles ou qui sont contenues par cette classe de conteneurs. En outre, comme une classe de piles ou de conteneurs ne peut être associée qu'à une seule classe de dispositifs d'enclos et que celle-ci peut être héritée des super classes de piles ou de conteneurs, ces dernières ne peuvent en posséder une si leurs sous-classes en disposent déjà.

Un parallélisme existe ici aussi entre la classe d'attributs composants associant une classe de piles à une classe de parts et la classe d'attributs dispositifs d'enclos associant la même classe de parts à la classe de dispositifs d'enclos de cette classe de piles. En effet, comme chaque pile ne possède qu'un seul dispositif d'enclos, celui-ci est défini par toutes les parts encloses de la pile. À l'échelle des classes, les cardinalités d'une classe d'attributs composants et d'une classe d'attributs dispositifs d'enclos qui possèdent la même classe de parts comme destination doivent être identiques ou adaptées au type de piles (FIFO, LIFO ou standard). Si la pile est de type FIFO ou LIFO, les bornes de la cardinalité de la classe d'attributs dispositifs d'enclos peuvent être inférieures d'une unité par rapport aux bornes de la cardinalité de la classe d'attributs composants. Si la pile est ouverte aux deux extrémités, les bornes peuvent être inférieures de deux unités. Cette règle reste d'application même si les cas de figure sont généralement plus complexes en raison du principe d'héritage des classes d'attributs.

Selon la cardinalité de la classe d'attributs composants associant une classe de piles à une classe de parts, le type de classes de piles et la modélisation choisie par l'utilisateur, une classe de parts composant une classe de piles ne doit pas nécessairement définir la classe de dispositifs d'enclos de cette classe de piles. Un attribut dispositif d'enclos n'est en effet pas nécessaire pour les instances d'une telle classe de parts composantes si celles-ci sont toujours situées à une extrémité ouverte de la pile. Sans tenir compte des choix de modélisation, c'est le cas si une pile est notamment composée de maximum une seule instance d'une classe de parts ou de maximum deux instances à condition que cette pile soit de type standard.

Un conteneur ne possède également qu'un seul dispositif d'enclos qui est défini par toutes les parts encloses de ce conteneur. Nous pouvons donc faire aussi un parallélisme entre la classe d'attributs contenants associant une classe de conteneurs à une classe de parts et la classe d'attributs dispositifs d'enclos associant la même classe de parts à la classe de dispositifs d'enclos de cette classe de conteneurs. Ne sachant pas quelle quantité de parts sont encloses, nous pouvons seulement affirmer que les bornes de la cardinalité de la classe d'attributs dispositifs d'enclos ne peuvent pas être supérieures aux bornes de la cardinalité de la classe d'attributs contenants.

Lors de la traduction du modèle graphique de "parts" en langage AlbertII, le projet VBA vérifie pour chaque classe de piles que chacune de ses classes de parts composantes soit associée – si nécessaire et même par héritage – à sa classe de dispositifs d'enclos, que sa classe de dispositifs d'enclos ne soit associée –même par héritage – qu'à ses classes de parts composantes, et que les cardinalités des classes d'attributs dispositifs d'enclos de sa classe de dispositifs d'enclos respectent ce qui a été énoncé précédemment. Le projet VBA vérifie également pour chaque classe de conteneurs que sa classe de dispositifs d'enclos ne soit associée – même par héritage – qu'à ses classes de parts contenues, et que les cardinalités des

classes d'attributs dispositifs d'enclos de sa classe de dispositifs d'enclos respectent le principe décrit ci-dessus.

En résumer, les parallélismes observés dans les deux dernières contraintes établissent un rapport d'interdépendance entre les trois chemins partant d'une même classe de piles ou de conteneurs et aboutissant à la même classe de parts composantes ou contenues soit directement, soit en passant par une classe de dispositifs de contiguïté, soit en passant par une classe de dispositifs d'enclos, ce dernier chemin étant parfois inexistant.

21. Le caractère exclusif des deux types de classes spéciales de parts composées

Selon la quatorzième contrainte sur les classes spéciales, une part composée ne peut être simultanément une pile et un conteneur.

Cette contrainte est vérifiée par la configuration de la modélisation graphique des "parts" parce que le gabarit "Gabarit_Pile_Conteneur_Tampon" du modèle de dessin "Systèmes manufacturiers" impose une forme différente pour créer (spécialiser) une classe de piles ou de conteneurs, instance de la méta-classe de parts composées, qui constitue automatiquement une sous-classe respective de la classe spéciale de piles ou de conteneurs.

22. Le nombre de composants d'une part composée et d'une pile et le nombre de contenus d'un conteneur

La cardinalité de la classe générale d'attributs composants ("2:N") exige qu'une part composée ou qu'une pile soit constituée d'au moins deux autres parts et qu'un conteneur contienne au moins deux autres parts. Cette contrainte fait l'objet d'une traduction en AlbertII. Toutefois, le projet VBA teste si cette contrainte peut être vérifiée en examinant les cardinalités des classes d'attributs composants et contenants attachées à chaque classe de parts, soit directement, soit par l'intermédiaire de ses super classes : la somme des bornes supérieures de ces cardinalités doit être supérieure ou égale à deux. Seules sont contrôlées les classes de parts composées, de piles ou de conteneurs qui ne font pas partie d'une hiérarchie de spécialisation ou qui en forment les feuilles. Les super classes sont en effet des classes abstraites et le type d'une super classe racine est déterminé par ses sous-classes feuilles.

La figure 3.22 montre le type de message envoyé à l'utilisateur lorsque cette contrainte n'est pas vérifiée. L'exemple est emprunté à celui de la modélisation du tracteur Massey Ferguson en présupposant que la classe de parts composées *Tracteur* n'est associée à aucune classe de parts composantes.

Figure 3.22 : Exemple de message envoyé lorsqu'une classe de parts composées

n'est associée à aucune classe de parts composantes (contrainte n° 22)

23. **Le nombre de parts composantes d'un dispositif géométrique**

La cardinalité de la classe générale d'attributs dispositifs géométriques ("2:N") impose qu'un dispositif géométrique soit défini par au moins deux parts composantes. Cette contrainte, dite de géométrie, fait l'objet d'une traduction en AlbertII. Cependant, le projet VBA vérifie si un dispositif géométrique peut être associé à plusieurs composants en examinant les cardinalités des classes d'attributs dispositifs géométriques attachées à chaque classe de dispositifs géométriques : la somme des bornes supérieures de ces cardinalités doit être supérieure ou égale à deux.

À l'inverse, la borne supérieure de la cardinalité d'une classe d'attributs dispositifs géométriques ne peut être plus élevée que la borne supérieure de la cardinalité de la classe d'attributs composants dont la destination est la même que celle de la classe d'attributs dispositifs géométriques. En effet, si une part composée est constituée de maximum cinq composants, un dispositif géométrique ne peut assembler plus de cinq composants de cette part composée. Dans la pratique, la contrainte de cardinalité qui s'exerce sur la classe d'attributs composants empêchera un dispositif géométrique d'assembler plus de composants que la limite maximale autorisée par la cardinalité de cette classe d'attributs. Toutefois, pour éviter d'induire l'utilisateur en erreur (lui laisser croire qu'un dispositif géométrique pourra assembler autant de composants), le projet VBA vérifie la valeur des bornes supérieures des classes d'attributs composants et dispositifs géométriques "liées" lors de la traduction du modèle graphique de "parts" en langage AlbertII.

24. **La syntaxe des classes de parts et d'états**

	Origine	**Destination**
Classe de parts **(+classe de tampons exceptée)** **(*classe de conteneurs exceptée)**	identités : 1 ou + positions : 0 ou + composants[33] : 1 ou + contenus[34] : 1 ou + contenants[35] : 1 ou + sous-classes : 0 ou 1 dispositifs physiques : 0 ou + géométries[36] : 0 ou + contiguïtés[37] : 1 ou + enclos[38] : 0 ou 1	composants*+ : 0 ou + contenus* : 0 ou + contenants* : 0 ou + sous-classes : 0 ou + disp. géométriq.* + : 0 ou + dispositifs contiguïté* : 0 ou + dispositifs d'enclos* : 0 ou +
Classe d'identités	—	identités : 1
Classe de positions	—	positions : 1
Classe de dispositifs physiques	valeurs : 0 ou +	dispositifs physiques : 1 ou +
Classe de dispositifs géométriq.	dispositifs géométriq. : 1 ou + valeurs : 0 ou +	dispositifs de fixation : 0 ou + géométries : 1

[33] Classe de parts composées et classe de piles uniquement.
[34] Classe de tampons uniquement.
[35] Classe de conteneurs uniquement.
[36] Classe de parts composées uniquement.
[37] Classe de piles et de conteneurs uniquement.
[38] Classe de piles et de conteneurs uniquement.

Classe de dispositifs de fixation	dispositifs de fixation : 1 ou + valeurs : 0 ou +	—
Classe de dispositifs contiguïté	dispositifs contiguïté : 1 ou 2 valeurs : 0 ou +	contiguïtés : 1
Classe de dispositifs d'enclos	dispositifs d'enclos : 1 ou + valeurs : 0 ou +	enclos : 1
Classe de dispositifs de valeur	—	valeurs : 1 ou +

Tableau 3.3 : Syntaxe des classes de parts et d'états

Les graphes des méta-classes et des classes spéciales du sous-ensemble du pattern *Part* déterminent de manière très stricte le type de classes d'attributs qui peut être affecté à chaque classe de parts ou d'états en fonction de la méta-classe dont elle est l'instance et de la classe spéciale dont elle est la spécialisation. En outre, le type de classes d'attributs peut avoir un caractère obligatoire ou facultatif et le nombre de classes appartenant à ce type peut faire l'objet d'une limitation au niveau de chaque classe de parts ou d'états.

Le tableau 3.3 indique le type et le nombre de classes d'attributs dont chaque type de classes de parts ou d'états peut/doit être l'origine ou la destination soit directement, soit par héritage. Ce tableau résulte des contraintes citées précédemment. La colonne "Origine" reflète notamment les contraintes de cardinalités présentées dans le graphe des méta-classes du sous-ensemble du pattern *Part* à la figure 3.3. Ces contraintes de cardinalité sont également indiquées à côté du nom des méta-classes d'attributs et des classes spéciales d'attributs associées aux formes de base des deux gabarits situés dans l'interface utilisateur du logiciel Visio. Au niveau de la colonne "Destination", le lecteur attentif observera qu'une classe d'états doit toujours être la destination d'au moins une classe d'attributs, tandis qu'une classe de parts ne l'est pas nécessairement. Les classes de dispositifs de fixation constituent une exception parce que, contrairement aux autres classes d'états, elles ne définissent pas de classes de parts : elles sont autonomes comme ces dernières.

Chaque classe d'identités et chaque classe de positions doit être associée à une et une seule classe de parts. Cette restriction permet de simplifier la traduction en AlbertII du caractère unique de l'identité et de la position d'une part. Elle limite en effet l'expression de cette contrainte à une seule classe de parts. Si plusieurs classes de parts étaient associées aux mêmes classes d'identités ou de positions, ainsi qu'à d'autres classes d'identités ou de positions différentes, la conception de la contrainte en langage AlbertII devrait prendre en considération toutes les configurations possibles et devrait en conséquence éclater cette contrainte en sous-contraintes correspondant chacune à une configuration différente.

Puisque la modélisation exige qu'un dispositif de contiguïté associe toujours deux parts, le nombre de classes d'attributs dispositifs de contiguïté qu'une classe de dispositifs de contiguïté peut posséder est limité à deux : une seule suffit si sa cardinalité équivaut à "2:2" et deux sont nécessaires si leurs cardinalités sont égales à "1:1".

Bien que les classes de dispositifs de contiguïté et d'enclos soient des instances de la méta-classe de dispositifs géométriques, elles ne peuvent définir une classe de dispositifs de fixation parce qu'elles sont des sous-ensembles de classes spéciales.

Le projet VBA vérifie la syntaxe des classes de parts et d'états au moment de la traduction du modèle graphique de "parts" en langage AlbertII. Dans une hiérarchie de spécialisation, conformément à ce qui a été mentionné au début du chapitre, il vérifie que les classes d'attributs identités et les éventuelles classes d'attributs positions de toutes les sous-classes soient uniquement attachées à la super classe racine. Il contrôle aussi que chaque classe de

parts mixtes soit la super classe directe d'au moins soit une classe de parts mixtes, soit deux classes de parts appartenant à des types différents et autres que le tampon et le conteneur.

25. **La syntaxe des classes d'attributs**

Les graphes des méta-classes et des classes spéciales du sous-ensemble du pattern *Part* déterminent strictement les types de classes de parts ou d'états qu'une classe d'attributs doit associer en fonction de la méta-classe dont elle est l'instance ou de la classe spéciale dont elle est la spécialisation. En outre, les bornes minimale et maximale autorisées pour la cardinalité d'une classe d'attributs sont déterminées par la cardinalité de la classe générale ou de la classe spéciale dont elle est la sous-classe.

Le tableau 3.4 indique le type des classes de parts et d'états qui sont l'origine et la destination de chaque type de classes d'attributs. Il signale également si la cardinalité de la classe d'attributs fait l'objet d'une vérification syntaxique.

	Origine	**Destination**	**Cardinalité**
Classe d'at. composants	classe de piles ou de parts composées	classe de parts*+	oui
Classe d'at. contenus	classe de tampons	classe de parts*	oui
Classe d'at. contenants	classe de conteneurs	classe de parts*	oui
Classe d'at. identités	classe de parts	classe d'identités	non
Classe d'at. positions	classe de parts	classe de positions	non
Classe d'at. dispositifs physiques	classe de parts	classe de dispositifs physiques	oui
Classe d'at. géométries	classe de parts composées	classe de dispositifs géométriques	oui
Classe d'at. dispositifs géométriques	classe de dispositifs géométriques	classe de parts composantes*+	oui
Classe d'at. dispositifs de fixation	classe de dispositifs de fixation	classe de dispositifs géométriques	oui
Classe d'at. dispositifs de contiguïté	classe de dispositifs de contiguïté	classe de parts*	non
Classe d'at. dispositifs d'enclos	classe de dispositifs d'enclos	classe de parts*	oui
Classe d'at. contiguïtés	classe de piles ou de conteneurs	classe de dispositifs de contiguïté	oui
Classe d'at. enclos	classe de piles ou de conteneurs	classe de dispositifs d'enclos	non
Classe d'at. valeurs	classe de dispositifs#	classe de dispositifs de valeur	oui
Classe d'at. sous-classes	classe de parts	classe de parts	non

+classe de tampons exceptée
***classe de conteneurs exceptée**
#classe de dispositifs de valeur exceptée

Tableau 3.4 : Syntaxe des classes d'attributs

La vérification syntaxique consiste à s'assurer que :

- la borne inférieure est un entier positif;
- la borne supérieure est un nombre indéfini[39] ("N" ou "n") ou un entier positif non nul;
- si les deux bornes sont des nombres, la borne inférieure n'est pas strictement plus grande que la borne supérieure.

Les cinq classes d'attributs dont la cardinalité ne fait pas l'objet d'un examen sont :

- les classes d'attributs identités et enclos dont la cardinalité est fixée à "1:1";
- la classe d'attributs dispositifs de contiguïté dont la cardinalité est égale à "1:1" ou à "2:2";
- la classe d'attributs positions dont la cardinalité est égale à "0:1" ou à "1:1";
- la classe d'attributs sous-classes qui ne possède pas de cardinalité.

La classe d'attributs valeurs est un peu particulière : d'une part parce que la borne inférieure de sa cardinalité doit être un entier positif non nul et, d'autre part, parce qu'elle contient en plus des informations sur la classe de dispositifs de valeur. Ces informations permettent à l'utilisateur d'indiquer si les valeurs (instances) de cette classe sont dupliquées et ordonnées. Le tableau 3.5 résume les caractéristiques possibles des valeurs d'un dispositif.

	Duplication	**Ordre**
SET (ensemble)	non	non
BAG (sac)	oui	non
SEQ (séquence)	oui	oui

Tableau 3.5 : Caractéristiques possibles des valeurs d'un dispositif

Un ensemble (*SET*) contient des valeurs uniques et non ordonnées. Un sac (*BAG*) renferme des valeurs qui peuvent être identiques mais qui ne sont pas ordonnées. Une séquence (*SEQ*) est constituée de valeurs ordonnées qui peuvent être dupliquées. Tous les trois peuvent être vides. Un dispositif ne peut contenir des valeurs à la fois uniques et ordonnées. La duplication et l'ordre sont associés à deux propriétés personnalisées représentées par des booléens. Lorsque l'utilisateur choisit l'option d'ordre (ordre à "VRAI"), le booléen de la duplication se met automatiquement à "VRAI". À l'instar de la contrainte sur l'unicité du nom des classes, cette procédure est exécutée "en temps réel" grâce à l'événement lié à la modification de la formule d'une cellule de la feuille ShapeSheet.

Lors de la traduction du modèle graphique de "parts" en langage AlbertII, le projet VBA vérifie que la cardinalité de la classe d'attributs valeurs permette la duplication des valeurs des dispositifs si celle-ci est demandée par l'utilisateur. À cette fin, il contrôle que la borne supérieure de la cardinalité soit strictement supérieure à un.

Les classes d'attributs contenus, composants et contenants sont également particulières parce qu'elles sont sémantiquement très proches (les deux dernières sont d'ailleurs des instances de la même méta-classe d'attributs) et parce que les contraintes sont différentes suivant les classes d'origine et de destination.

Le tableau 3.6 résume les compositions et les contenances possibles entre les classes de parts. Pour chaque ligne du tableau, un booléen indique si le type de classes de parts peut être le composé ou le contenant d'un type de classes de parts situé dans une des colonnes. Inversement, pour chaque colonne du tableau, un booléen indique si le type de classes de parts peut être le composant ou le contenu d'un type de classes de parts placé dans une des lignes.

[39] Le nombre indéfini "N" ou "n" représente toujours n'importe quel entier positif non nul.

Nous avons vu que, par définition, les parts de base et les parts mixtes ne pouvaient pas être composées d'autres parts. Par contre, les piles et les parts composées peuvent être composées de n'importe quel type de parts sauf des tampons et des conteneurs parce que les premiers possèdent une position fixe tandis que les seconds constituent des entités virtuelles. Ces deux derniers types peuvent contenir tous les types de parts sauf des conteneurs pour la même raison que celle invoquée ci-dessus. Un tampon ne peut pas non plus servir de composant parce qu'il n'est pas une véritable part au sens manufacturier du terme : il est plutôt défini comme un endroit où un certain nombre de parts peuvent demeurer quelque temps. Nous permettons toutefois à un tampon de contenir d'autres tampons parce qu'un tampon qui serait subdivisé en plusieurs parties pourrait être modélisé au moyen d'un tampon qui contiendrait autant de tampons que de parties. D'autre part, dans la réalité, un tampon peut se présenter sous la forme d'un container et, par voie de conséquence, être contigu aux parts qu'il contient, voire même les enclore. Nous laissons donc la possibilité à l'utilisateur d'inclure des tampons dans les conteneurs afin d'exercer des contraintes d'accessibilité sur ces tampons.

La classe d'attributs sous-classes est aussi particulière parce que les contraintes sont différentes selon les classes d'origine et de destination. Voir à ce sujet le tableau 3.1 résumant la spécialisation des classes de parts.

contient / est constituée de	**Part de base**	**Part composée**	**Part mixte**	**Pile**	**Conteneur**	**Tampon**
Part de base	F	F	F	F	F	F
Part composée	V	V	V	V	F	F
Part mixte	F	F	F	F	F	F
Pile	V	V	V	V	F	F
Conteneur	V	V	V	V	F	V
Tampon	V	V	V	V	F	V

Tableau 3.6 : Composition et contenance des classes de parts

Le projet VBA vérifie la syntaxe des classes d'attributs au moment de la traduction du modèle graphique de "parts" en langage AlbertII.

26. **La syntaxe des commentaires**

Lors de la traduction du modèle graphique de "parts" en langage AlbertII, le projet VBA vérifie que chaque commentaire soit attaché à une classe de parts ou d'états. Il examine simplement si le commentaire n'est pas isolé car il est impossible de l'associer à une classe d'attributs puisque la forme qui le représente ne peut être fixée qu'à une forme bidimensionnelle.

27. **Le nombre de dispositifs géométriques d'un dispositif de fixation**

La cardinalité de la classe générale d'attributs dispositifs de fixation ("1:N") requiert que chaque dispositif de fixation soit défini par au moins un dispositif géométrique. Cette contrainte, dite de fixation, fait l'objet d'une traduction en AlbertII. Toutefois, contrairement aux classes de parts composées, de piles, de conteneurs et de dispositifs géométriques qui font

l'objet des contraintes n° 22 et 23, le projet VBA ne vérifie pas si un dispositif de fixation peut être associé à un dispositif géométrique car cette exigence est vérifiée par la combinaison de la contrainte n° 24 sur la syntaxe des classes de parts et d'états qui stipule qu'une classe de dispositifs de fixation doit toujours être associée à au moins une classe de dispositifs géométriques et de la contrainte n° 25 sur la syntaxe des classes d'attributs qui précise que la borne supérieure de la cardinalité d'une classe d'attributs ne peut jamais être nulle. En conséquence, la somme des bornes supérieures des cardinalités des classes d'attributs dispositifs de fixation d'une classe de dispositifs de fixation est toujours supérieure ou égale à un.

Un parallélisme existe entre les cardinalités des classes d'attributs géométries et dispositifs de fixation qui possèdent la même classe de dispositifs géométriques comme destination. Pour illustrer notre propos, prenons l'exemple d'un parallélépipède rectangle composé de quatre rectangles et de deux carrés situés aux extrémités. Lorsqu'il est complet, le parallélépipède rectangle nécessite l'utilisation de six dispositifs géométriques : quatre pour unir les rectangles deux à deux et deux pour joindre chaque carré à l'un des petits côtés de chacun des quatre rectangles. Si tous les éléments du parallélépipède rectangle sont soudés, nous avons besoin d'un seul dispositif de fixation défini par les six dispositifs géométriques, instances de deux classes de dispositifs géométriques différentes.

Logiquement, la somme des bornes supérieures des cardinalités des deux classes d'attributs géométries ou dispositifs de fixation qui relient la classe de parts composées ou de dispositifs de fixation aux deux classes de dispositifs géométriques doit être égale à six. Mais plusieurs dispositifs de fixation pourraient fixer chacun l'ensemble ou une partie des dispositifs géométriques. La vérification de la validité des cardinalités des classes d'attributs dispositifs de fixation fait donc davantage appel à la sémantique qu'à la syntaxe car elle dépend des choix de modélisation faits par l'utilisateur. Elle ne peut être entièrement effectuée par le projet VBA : celui-ci vérifie si les bornes supérieures/inférieures des cardinalités des classes d'attributs dispositifs de fixation ne sont pas plus élevées que les bornes supérieures/inférieures des cardinalités des classes d'attributs géométries mais la conception correcte du modèle repose toujours sur l'utilisateur.

Si les bornes supérieures des cardinalités sont trop élevées, ce qui est vérifié par le projet VBA, cela n'aura pas d'incidence dans la fabrication des parts car cela n'influencera pas le nombre de dispositifs géométriques. Par contre, si elles sont inférieures au nombre de dispositifs géométriques prévus, cela risque de poser problème dans l'assemblage des composants. Dans notre exemple, si un dispositif de fixation admet moins de six dispositifs géométriques, aucun parallélépipède rectangle ne sera jamais complet. Toutefois, comme nous l'avons déjà fait remarquer, le concepteur est aussi responsable du fonctionnement correct du modèle d'assemblage des parts.

28. **Chaque classe d'attributs associe deux classes de parts ou d'états différentes**

En pratique, cela signifie qu'une relation ne peut pas associer deux fois la même forme. Cette éventualité est prise en compte par le projet VBA bien qu'elle contrevienne automatiquement à l'une des contraintes citées précédemment. Elle requiert en effet l'association de deux classes du même type. Or, vu la contrainte n° 25 relative à la syntaxe des classes d'attributs, seules les relations composants, contenus, contenants et sous-ensembles peuvent réunir deux instances d'une même méta-classe. Mais dans ce cas, elles entrent en contradiction, d'une part, avec la contrainte n° 5 (aucune part ni aucune pile n'est composée d'elle-même), la contrainte n° 14 (aucun tampon ne se contient lui-même) et la

contrainte n° 13 (la hiérarchisation des sous-classes de parts) parce qu'elles forment un circuit et, d'autre part, avec la propriété du conteneur selon laquelle un conteneur ne peut être ni un composant, ni un contenu (en d'autres termes, il ne peut être la destination d'une relation composant, contenu ou contenant).

29. **L'ajout et la suppression de pages**

Les algorithmes utilisés dans le projet VBA nécessitent la numérotation continue des pages (feuilles) en débutant à un. À cette fin, nous avons conçu deux macros accessibles grâce à deux boutons situés sur la barre d'outils du logiciel Visio. La première ajoute une feuille au projet en lui attribuant automatiquement le numéro qui suit celui de la page active et en re-numérotant les pages suivantes afin de conserver une pagination continue. La seconde supprime la feuille active et met à jour la numérotation comme la macro précédente. Cette suppression ne peut se réaliser que si la feuille est vide. En effet, nous attribuons à chaque classe de parts et d'états lors de leur création un numéro unique qui est employé lorsque le modèle graphique de "parts" est traduit en langage AlbertII : il sert d'indice au tableau temporaire contenant les données nécessaires à la traduction des classes en types. Un tableau permanent contient les informations relatives à cette numérotation et, lorsqu'une forme est supprimée, un gestionnaire d'événements provoque l'exécution d'un code qui met à jour ce tableau. Or, la suppression d'une feuille contenant des formes ne déclenche pas cet événement. Par conséquent, nous exigeons que toutes les formes présentes sur une page soient effacées manuellement par l'utilisateur avant la suppression de cette page.

D'autre part, si le modèle ne contient plus qu'une seule page, celle-ci ne peut pas être supprimée. En effet, lorsque nous effaçons toutes les pages d'un projet, le logiciel Visio en ajoute automatiquement une en lui attribuant comme nom le mot "page" suivi d'un numéro aléatoire. Or, nous souhaitons précisément que la dénomination d'une page corresponde à sa position dans la pagination, soit ici, en l'occurrence, que la nouvelle page reçoive l'appellation suivante : "1".

30. **Un modèle graphique de "parts" ne peut être vide**

Lors de la traduction du modèle graphique de "parts" en langage AlbertII, le projet VBA vérifie que le modèle contient au moins une classe (forme). Nous partons en effet du principe que la traduction d'un modèle vide est absurde. Plus précisément, tout modèle doit posséder au moins une classe de parts. En pratique, cette contrainte particulière n'est pas testée car un modèle non vide qui ne contient pas de classes de parts contrevient au minimum à l'une des nombreuses contraintes citées précédemment et principalement aux contraintes n° 24 et 25 sur la syntaxe des classes de parts, d'états et d'attributs.

3.1.4.2 Les contraintes traduites en AlbertII

Les contraintes qui ne peuvent pas être vérifiées par le projet sont traduites en AlbertII pour être associées aux classes de parts et d'états auxquelles elles se rapportent.[40] Les options que nous avons prises dans la réalisation du modèle de dessin "Systèmes manufacturiers" et les modifications que nous avons opérées par rapport au pattern *Part* tel que décrit dans la thèse nous ont permis d'éviter le recours à certaines contraintes AlbertII. La première de ces contraintes est liée à la classe des parts existantes : si une part existe, soit elle ne possède pas de composants parce que son type est "BasicPart", soit elle est constituée d'au moins deux composants parce que son type est "CompoundPart". Les deuxième et troisième contraintes AlbertII rendues inutiles sont associées à chaque classe de parts : elles indiquent, pour l'une, le type de parts ("BasicPart" ou "CompoundPart") et, pour l'autre, le fait que toutes les parts sont des parts existantes (elles possèdent donc une identité dont la valeur n'est pas indéfinie).

Les huit contraintes traduites en AlbertII sont respectivement intitulées comme suit :

- la contrainte d'identité;
- la contrainte de composition;
- la contrainte de contenance;
- la contrainte de cardinalité;
- la contrainte de contiguïté;
- la contrainte d'énumération;
- la contrainte de géométrie;
- la contrainte de fixation.

31. **Chaque part possède une identité distincte**

La contrainte n° 7 présentée à la section précédente vérifie que chaque part possède une identité. Quant au caractère unique de cette identité, il fait l'objet d'une contrainte rédigée en langage AlbertII.

Une classe de parts peut être associée à plusieurs classes d'identités. Chaque part est alors liée à une instance de chacune des classes identités qui sont unies à la classe de parts dont elle est l'instance. Dans la thèse, chaque instance d'une classe d'identités (un identifiant) doit être unique. [Petit 1999, p. 185] Or, nous estimons que ce principe est trop restrictif. Nous pensons en effet que deux parts, instances d'une même classe, possèdent une identité différente si elles sont définies par deux instances différentes d'une seule de leurs classes d'identités. Cette amélioration est utile dans le cas, par exemple, d'un utilisateur qui souhaiterait prendre l'année de fabrication comme classe d'identités car, jusqu'à présent, il ne pouvait pas le faire à moins de fabriquer une part par année. Toute classe d'identités étant associée à une et une seule classe de parts, la rédaction de cette contrainte en est facilitée car elle ne doit pas prendre en considération plusieurs classes de parts.

```
%BASIC TYPES
Année_production
Numéro_usine
Numéro_national
%CONSTRUCTED TYPES
```

[40] Pour un exemple de modélisation graphique de "parts" avec une traduction des contraintes en AlbertII, voir l'annexe D.

```
//CLASSES DE PARTS ÉLÉMENTAIRES
Moteur = CP [ type:BasicPart,
              identité1:Année_production,
              identité2:Numéro_usine,
              identité3:Numéro_national]
      //Contrainte d'identité
      \WITH \ForAll p/Moteur \and \ForAll q/Moteur :
            p <> q => identité1(p) <> identité1(q) \or
                      identité2(p) <> identité2(q) \or
                      identité3(p) <> identité3(q)
```

Figure 3.23 : Exemple d'une contrainte AlbertII sur l'unicité de l'identité des parts

La figure 3.23 montre un exemple de contrainte relative à l'unicité de l'identité des parts (moteurs) d'une classe de parts *Moteur* associée à trois classes d'identités faisant référence à l'année de production (*Année_production*), au numéro d'usine (*Numéro_usine*) et à un numéro national (*Numéro_national*) composé de lettres et de chiffres. Cette liaison est assurée par trois classes d'attributs identités intitulées *identité1*, *identité2* et *identité3*. Cette contrainte est précédée de la traduction de la classe de parts et des trois classes d'identités en types de données de base et en types de données construits AlbertII.

32. **Une part composée ou une pile est constituée d'au moins deux autres parts**

La contrainte n° 22 vérifie que les instances de chaque classe de parts composées ou de piles du modèle peuvent être constituées d'au moins deux parts en examinant la cardinalité de leurs classes d'attributs composants. Lors de la traduction d'une classe de parts composées, le projet VBA détermine si la rédaction de la contrainte de composition est nécessaire. Celle-ci est en effet vérifiée si la somme des bornes inférieures des cardinalités des classes d'attributs composants associées à cette classe est supérieure ou égale à deux. Dans le cas contraire, elle doit être exprimée en AlbertII. Si la somme mentionnée équivaut à zéro, nous exigeons que le nombre d'attributs composants de chaque part composée ou de chaque pile soit égal ou supérieur à deux. Si elle est égale à un, nous requérons pour chaque part composée ou pour chaque pile soit que le nombre d'attributs composants qui sont les instances de la classe d'attributs composants dont la borne inférieure de la cardinalité est égale à un soit égal ou supérieur à deux (si cela est possible), soit que le nombre d'attributs composants de n'importe quelle autre classe d'attributs composants soit égal ou supérieur à un (si cela est possible).

Nous avons choisi d'effectuer une différence de traitement en fonction de la valeur de cette somme pour une raison purement esthétique (éviter l'uniformité et donc la monotonie des contraintes) car la facilité nous aurait commandé d'appliquer le même traitement global à toutes les classes de parts composées ou de piles, quelle qu'ait été la cardinalité de leurs classes d'attributs composants. Dans le cas où la somme équivalait à zéro, nous n'avons pas traité les classes d'attributs composants individuellement car nous aurions dû prendre en considération toutes les configurations possibles et celles-ci étaient trop nombreuses (car exponentielles).

```
%CONSTRUCTED TYPES
//CLASSES DE PARTS ÉLÉMENTAIRES
Bloc_moteur = CP [ type:CompoundPart,
                   composant_bloc:Bloc*,
                   composant_pistons:SET[Piston],
                   composant_cylindres:SET[Cylindre]]
      //Contrainte de composition
```

```
      \WITH \ForAll p/Bloc_moteur :
            (Card(composant_bloc) +
            Card(composant_pistons) +
            Card(composant_cylindres)) >= 2
Voiture = CP [ type:CompoundPart,
                composant_châssis:Châssis,
                composant_bloc_moteur:Bloc_moteur*,
                composant_pneus:SET[Pneu]]
      //Contrainte de composition
      \WITH \ForAll p/Voiture :
            Card(composant_bloc_moteur) = 1 \or
            Card(composant_pneus) >= 1
```

Figure 3.24 : Exemple d'une contrainte AlbertII sur les parts composées

La figure 3.24 illustre par un exemple la contrainte de composition. Celle-ci s'exerce sur une classe de parts composées *Bloc_moteur* constituée d'un à zéro bloc (*Bloc*), de zéro à quatre pistons (*Piston*) et de zéro à quatre cylindres (*Cylindre*), ainsi que sur une classe de parts composées *Voiture* constituées d'un châssis (*Châssis*), d'un ou zéro bloc moteur (*Bloc_moteur*) et de zéro à quatre pneus (*Pneu*). Les classes de parts composantes sont respectivement associées aux deux classes de parts composées par les classes d'attributs *composant_bloc*, *composant_pistons*, *composant_cylindres*, *composant_châssis*, *composant_bloc_moteur* et *composant_pneus*. Cet exemple ne montre que les contraintes et les parties de types construits relatives à la composition. La fonction "*Card*" utilisée ici est définie dans la section suivante consacrée à la définition d'un sous-langage AlbertII pour le sous-ensemble du pattern *Part*.

33. **Un conteneur contient au moins deux parts**

La contrainte n° 22 vérifie que les instances de chaque classe de conteneurs du modèle graphique de "parts" peuvent contenir au moins deux parts en examinant la cardinalité de leurs classes d'attributs contenants. Lors de la traduction d'une classe de conteneurs en langage AlbertII, le projet VBA détermine si la rédaction de la contrainte de contenance est nécessaire. Celle-ci est en effet vérifiée si la somme des bornes inférieures des cardinalités des classes d'attributs contenants associées à cette classe est supérieure ou égale à deux. Dans le cas contraire, elle doit être exprimée en AlbertII. Pour les mêmes raisons que celles citées à la contrainte de composition, nous appliquons un traitement individuel aux attributs contenants si la somme mentionnée précédemment équivaut à un et un traitement global si elle équivaut à zéro.

```
%CONSTRUCTED TYPES
//CLASSES DE PARTS ÉLÉMENTAIRES
Conteneur_bloc_moteur = CP [ type:Container,
                             contenant_bloc:Bloc*,
                             contenant_pistons:SET[Piston],
                             contenant_cylindres:SET[Cylindre]]
      //Contrainte de contenance
      \WITH \ForAll p/Conteneur_bloc_moteur :
            (Card(contenant_bloc) +
            Card(contenant_pistons) +
            Card(contenant_cylindres)) >= 2
Conteneur_voiture = CP [ type:Container,
                         contenant_châssis:Châssis,
```

```
                        contenant_bloc_moteur:Bloc_moteur*,
                        contenant_pneus:SET[Pneu]]
      //Contrainte de contenance
      \WITH \ForAll p/Conteneur_voiture :
            Card(contenant_bloc_moteur) = 1 \or
            Card(contenant_pneus) >= 1
```

Figure 3.25 : Exemple d'une contrainte AlbertII sur les conteneurs

La figure 3.25 illustre la contrainte de contenance par un exemple inspiré de celui décrit à la contrainte précédente relative à la composition. Soit la classe de conteneurs *Conteneur_Bloc_moteur* contenant les composants de la classe *Bloc_moteur* avec des cardinalités identiques à celles des classes d'attributs composants et indiquant que les pistons et les cylindres sont contigus entre eux et enclos dans le bloc. Soit la classe de conteneurs *Conteneur_Voiture* contenant les composants de la classe *Voiture* avec des cardinalités identiques à celles des classes d'attributs composants et indiquant que le châssis est contigu au bloc moteur et aux pneus. Les classes de parts contenues sont respectivement associées aux deux classes de conteneurs au moyen des classes d'attributs *contenant_bloc*, *contenant_pistons*, *contenant_cylindres*, *contenant_bloc_moteur*, *contenant_châssis* et *contenant_pneus*. Cet exemple ne montre que les contraintes et les parties de types construits relatives à la contenance.

34. **La contrainte de cardinalité**

La cardinalité des classes d'attributs est vérifiée par le projet au niveau syntaxique mais elle ne peut pas être traduite directement dans les types de données AlbertII. Elle doit par conséquent faire l'objet d'une contrainte spécifique. Celle-ci consiste simplement à indiquer les cardinalités minimales et maximales autorisées.

```
%CONSTRUCTED TYPES
//CLASSES DE PARTS ÉLÉMENTAIRES
Bloc_moteur = CP [ type:CompoundPart,
                   composant_bloc:Bloc*,
                   composant_pistons:SET[Piston],
                   composant_cylindres:SET[Cylindre]]
      //Contrainte de cardinalité
      \WITH \ForAll p/Bloc_moteur :
            Card(composant_pistons) <= 4 \and
            Card(composant_cylindres) <= 4
Voiture = CP [ type:CompoundPart,
                composant_châssis:Châssis,
                composant_bloc_moteur:Bloc_moteur*,
                composant_pneus:SET[Pneu]
                dispositif_portes:SET[Porte]]
      //Contrainte de cardinalité
      \WITH \ForAll p/Voiture :
            Card(composant_pneus) <= 4 \and
```

```
        Card(dispositif_portes) >= 3 \and
        Card(dispositif_portes) <= 5
```

Figure 3.26 : Exemple d'une contrainte AlbertII sur la cardinalité des classes d'attributs

La figure 3.26 reprend l'exemple de la figure 3.24 pour illustrer la contrainte de cardinalité. Toutefois, nous associons à la classe de parts composées *Voiture* la classe de dispositifs physiques *Porte* qui indique si la voiture comporte trois, quatre ou cinq portes. Les deux classes sont réunies au moyen de la classe d'attributs dispositifs physiques *dispositif_portes* dont la cardinalité est fixée à "3:5".

35. **Une pile ou un conteneur est défini par au moins un dispositif de contiguïté**

La cardinalité de la classe spéciale d'attributs contiguïtés ("1:N") exige qu'une pile ou qu'un conteneur soit défini par au moins un dispositif de contiguïté. Toutefois, la réalisation de cette contrainte ne doit pas être vérifiée avant que celle-ci ne soit traduite en AlbertII car la combinaison de la contrainte n° 24 sur la syntaxe des classes de parts et d'états et de la contrainte n° 25 sur la syntaxe des classes d'attributs garantit qu'une classe de piles ou de conteneurs est toujours associée à au moins une classe de dispositifs de contiguïté au travers d'une classe d'attributs contiguïtés dont la borne supérieure est un entier positif non nul.

Lors de la traduction d'une classe de piles ou de conteneurs en langage AlbertII, le projet VBA détermine si la rédaction de la contrainte de contiguïté est nécessaire. Celle-ci est en effet vérifiée si la somme des bornes inférieures des cardinalités des classes d'attributs contiguïtés associées à cette classe n'est pas nulle. Dans le cas contraire, elle doit être exprimée en AlbertII. Nous appliquons un traitement individuel aux attributs contiguïtés.

```
%CONSTRUCTED TYPES
//CLASSES DE PARTS ELEMENTAIRES
Pièces_euros = CP [type:PileOfParts,
                   cont_2€:SET[Pièces_2€],
                   cont_1€:SET[Pièces_1€],
                   cont_euros:SET[Contiguïté_euros]]
     //Contrainte de contiguïté
     \WITH \ForAll p/Pièces_euros :
          Card(cont_euros) >= 1
```

Figure 3.27 : Exemple d'une contrainte AlbertII sur les dispositifs de contiguïté

La figure 3.27 illustre par un exemple la contrainte de contiguïté. Soit la classe de piles *Pièces_euros* composée des classes de pièces de deux euros (*Pièces_2€*) et d'un euro (*Pièces_1€*) au travers des classes d'attributs composants respectives *cont_2€* et *cont_1€* de cardinalité "0:N", et associée à la classe de dispositifs de contiguïté *Contiguïté_euros* au travers de la classe d'attributs contiguïtés *cont_euros* de cardinalité "0:N". Cet exemple ne montre que la contrainte et les parties de types construits relatives à la contiguïté.

36. **La contrainte d'énumération**

La contrainte d'énumération concerne uniquement la super classe racine d'une hiérarchie de spécialisation. Elle est liée au type de cette classe : chaque sous-classe feuille détermine les états qui peuvent être associés aux parts qui sont les instances de la sous-classe feuille et de ses super classes. En pratique, les classes d'attributs (héritées ou non) appartenant à la sous-classe feuille qui détermine le type de la super classe racine doivent renvoyer une valeur définie sur laquelle peuvent s'exercer des contraintes de composition, de contenance, de cardinalité ou de contiguïté, tandis que les autres classes d'attributs de la hiérarchie doivent renvoyer une valeur indéfinie.

Les classes d'attributs identités et positions ne sont pas concernées par cette contrainte car elles appartiennent uniquement à la super classe racine et, quel que soit le type de celle-ci, elles sont toujours héritées par les sous-classes feuilles. Les classes d'attributs composants, contenus, contenants, dispositifs physiques, géométries, contiguïtés et enclos sont les seules intéressées.

```
%BASIC TYPES
Couleur
Contrôle
Test
%CONSTRUCTED TYPES
//SUPER CLASSES DE PARTS RACINES
Bloc_moteur = CP[ type:ENUM[ Bloc_en_production,
                             Bloc_au_contrôle_phase1,
                             Bloc_au_contrôle_phase2,
                             Bloc_testé],
                  résultat_contrôles:SET[Contrôle],
                  résultat_test:Test*,
                  couleur_du_bloc:Couleur]
     //Contrainte d'énumération
     \WITH \ForAll p : Bloc_moteur
          (Type(p)=Bloc_en_production <=> résultat_test = \undef
               \and résultat_contrôles = \undef) \and
          (Type(p)=Bloc_au_contrôle_phase1 <=> résultat_contrôles
               <> \undef \and résultat_test = \undef \and
               Card(résultat_contrôles) <= 10) \and
          (Type(p)=Bloc_au_contrôle_phase2 <=> résultat_contrôles
               <> \undef \and résultat_test = \undef \and
               Card(résultat_contrôles) <= 10) \and
          (Type(p)=Bloc_testé <=> résultat_test <> \undef \and
               résultat_contrôles = \undef)
//CLASSES DE PARTS SOUS-ENSEMBLES ET SUPER CLASSES
Bloc_au_contrôle = Bloc_moteur
//SOUS-CLASSES DE PARTS FEUILLES
Bloc_en_production = CP[ type:BasicPart,
                         isA:Bloc_moteur]
Bloc_testé = CP[ type:BasicPart,
                 isA:Bloc_moteur]
Bloc_au_contrôle_phase1 = CP[ type:BasicPart,
                              isA:Bloc_au_contrôle]
Bloc_au_contrôle_phase2 = CP[ type:BasicPart,
                              isA:Bloc_au_contrôle]
```

Figure 3.28 : Exemple d'une contrainte AlbertII sur l'énumération
au sein d'une hiérarchie de spécialisation

La figure 3.28 montre un exemple de contrainte d'énumération emprunté à la production de moteurs. Ces derniers passent par trois phases qui sont la construction, le contrôle puis la sortie d'usine (fin des tests avec leur résultat). Ils sont accompagnés de caractéristiques permanentes qui leur sont propres ou temporaires qui sont particulières à ces trois phases. La hiérarchie de spécialisation est composée de six classes de parts de base dont trois sont associées à des classes de dispositifs physiques. La super classe racine *Bloc_moteur* est liée à la classe de dispositifs physiques *Couleur* via la classe d'attributs *couleur_du_bloc* de cardinalité "1:1". Elle est partitionnée en trois sous-classes : *Bloc_en_production*, *Bloc_au_contrôle* et *Bloc_testé*. La deuxième est reliée à la classe de dispositifs physiques *Contrôle* par la classe d'attributs *résultat_contrôles* de cardinalité "0:10". Elle se décompose en deux sous-classes : *Bloc_au_contrôle_phase1* et *Bloc_au_contrôle_phase2*. La sous-classe *Bloc_testé* est unie à la classe de dispositifs physiques *Test* grâce à la classe d'attributs *résultat_test* de cardinalité "1:1".

Sur la figure 3.28, le troisième champ du produit cartésien définissant le type de données construit *Bloc_moteur* indique que la valeur du type *Test* peut être indéfinie (en raison de la présence d'une étoile placée à la fin de son nom). Or, la borne minimale de la cardinalité de la relation *résultat_test* vaut un. L'explication réside dans le fait qu'un moteur ne possède pas le dispositif physique *Test* s'il est en production ou au contrôle. En d'autres termes, une instance de *Bloc_moteur* n'est pas associée à une instance de *Test* si elle est aussi une instance de *Bloc_en_production* ou de *Bloc_au_contrôle*. Le deuxième champ prend déjà en compte cette éventualité puisqu'un ensemble (*SET*) peut être vide. Quant au quatrième champ, la valeur du type *Couleur* ne peut jamais être indéfinie car la classe qu'il représente est associée à la super classe racine dont sont issues toutes les instances (moteurs) de la hiérarchie. Il n'intervient donc pas dans la contrainte d'énumération. Lorsqu'un moteur est de type *Bloc_au_contrôle*, la valeur du dispositif physique *Contrôle* est définie et une contrainte de cardinalité correspondant à la classe d'attributs dispositifs physiques *résultat_contrôles* s'exerce sur cette valeur.

37. **Un dispositif géométrique assemble au moins deux parts composantes**

La contrainte n° 23 vérifie que chaque instance d'une classe de dispositifs géométriques du modèle graphique de "parts" peut définir la manière dont sont assemblées au moins deux parts constituant les composants d'une part composante. Lors de la traduction d'une classe de dispositifs géométriques en langage AlbertII, le projet VBA détermine si la cardinalité de la classe générale d'attributs dispositifs géométriques ("2:N") doit faire l'objet d'une contrainte, dite de géométrie, rédigée en AlbertII. Celle-ci est en effet vérifiée si la somme des bornes inférieures des cardinalités des classes d'attributs dispositifs géométriques associées à cette classe est supérieure ou égale à deux. Pour les mêmes raisons que celles citées à la contrainte de composition, nous appliquons un traitement individuel aux attributs dispositifs géométriques si la somme mentionnée précédemment équivaut à un et un traitement global si elle équivaut à zéro.

Si une classe de dispositifs géométriques possède une seule classe d'attributs dispositifs géométriques, un dispositif géométrique qui est une instance de cette classe assemble des parts qui sont des instances de la même classe de parts composantes. Dans le cas contraire, ce dispositif géométrique peut lier des parts qui sont des instances de différentes classes de parts.

Si une classe de dispositifs géométriques possède une classe d'attributs dispositifs géométriques dont la borne supérieure de la cardinalité est équivalente à un, un dispositif

géométrique qui est une instance de cette classe ne peut pas assembler plusieurs instances de la classe de parts qui est la destination de cette classe d'attributs. Dans tous les autres cas, ce dispositif géométrique peut lier plusieurs instances de cette même classe de parts.

```
%CONSTRUCTED TYPES
//CLASSES DE PARTS ÉLÉMENTAIRES
Paquet_simple = CP[type:CompoundPart,
                   composant_simple:SET[Pot_pêche],
                   géo_pêche_simple:SET[Attache_pêche_simple]]
Paquet_mixte = CP[ type:CompoundPart,
                   composant_mixte_pêche:SET[Pot_pêche],
                   composant_mixte_ananas:SET[Pot_ananas],
                   géo_pêche_mixte:Attache_pêche_mixte,
                   géo_ananas_mixte:Attache_ananas_mixte,
                   géo_pêche_ananas:SET[Attache_pêche_ananas]]
//CLASSES DE DISPOSITIFS GÉOMÉTRIQUES
Attache_pêche_simple = CP[  type:PartGeometricalFeature,
                            attache_pêche:SET[Pot_pêche]]
Attache_pêche_mixte = CP[ type:PartGeometricalFeature,
                          attache_simple_pêche:SET[Pot_pêche]]
Attache_ananas_mixte = CP[  type:PartGeometricalFeature,
                            attache_simple_ananas:SET[Pot_ananas]]
     //Contrainte de géométrie
     \WITH \ForAll p/Attache_ananas_mixte :
          Card(attache_simple_ananas) = 2
Attache_pêche_ananas = CP[  type:PartGeometricalFeature,
                            attache_mixte_pêche:Pot_pêche*,
                            attache_mixte_ananas:Pot_ananas*]
     //Contrainte de géométrie
     \WITH \ForAll p/Attache_pêche_ananas :
          (Card(attache_mixte_pêche) +
          Card(attache_mixte_ananas)) >= 2
```

Figure 3.29 : Exemple d'une contrainte AlbertII sur les dispositifs géométriques

La figure 3.29 illustre la contrainte de géométrie par un exemple emprunté à une firme d'assemblage de pots de yogourts. Cette société fabrique des paquets simples composés de quatre pots de yogourts aux pêches et des paquets mixtes constitués de deux pots de yogourts aux pêches et de deux aux ananas. Chaque pot d'un paquet simple est attaché à deux pots aux pêches tandis que chaque pot d'un paquet mixte est attaché à deux pots de nature différente.

Le modèle graphique des paquets de yogourts est formé de la classe de parts composées *Paquet_simple* associée à la classe de parts de base (composantes) *Pot_pêche* par l'intermédiaire de la classe d'attributs composants *composant_simple* de cardinalité "4:4", ainsi que de la classe de parts composées *Paquet_mixte* associée aux deux classes de parts de base (composantes) *Pot_pêche* et *Pot_ananas* par l'intermédiaire, respectivement, des classes d'attributs composants *composant_mixte_pêche* et *composant_mixte_ananas* de cardinalités "2:2". Le modèle comprend également la classe de dispositifs géométriques *Attache_pêche_simple* associée à la classe de parts *Pot_pêche* par l'intermédiaire de la classe d'attributs dispositifs géométriques *attache_pêche* de cardinalité "2:2" et à la classe de parts composées *Paquet_simple* par l'intermédiaire de la classe d'attributs géométries *géo_pêche_simple* de cardinalité "4:4", la classe de dispositifs géométriques *Attache_pêche_mixte* associée à la

classe de parts *Pot_pêche* par l'intermédiaire de la classe d'attributs dispositifs géométriques *attache_simple_pêche* de cardinalité "2:2" et à la classe de parts composées *Paquet_mixte* par l'intermédiaire de la classe d'attributs géométries *géo_pêche_mixte* de cardinalité "1:1", la classe de dispositifs géométriques *Attache_ananas_mixte* associée à la classe de parts *Pot_ananas* par l'intermédiaire de la classe d'attributs dispositifs géométriques *attache_simple_ananas* de cardinalité "1:2" et à la classe de parts composées *Paquet_mixte* par l'intermédiaire de la classe d'attributs géométries *géo_ananas_mixte* de cardinalité "1:1", ainsi que de la classe de dispositifs géométriques *Attache_pêche_ananas* associée aux deux classes de parts *Pot_pêche* et *Pot_ananas* par l'intermédiaire, respectivement, des classes d'attributs dispositifs géométriques *attache_mixte_pêche* et *attache_mixte_ananas* de cardinalités "0:1" et à la classe de parts composées *Paquet_mixte* par l'intermédiaire de la classe d'attributs géométries *géo_pêche_ananas* de cardinalité "2:2". Cet exemple ne montre que les contraintes de géométrie et les parties de types construits qui y font référence.

38. **Un dispositif de fixation est défini par au moins un dispositif géométrique**

Comme nous l'avons indiqué à la contrainte n° 27, un dispositif de fixation peut toujours être associé à un dispositif géométrique car la somme des bornes supérieures des cardinalités des classes d'attributs dispositifs de fixation d'une classe de dispositifs de fixation est toujours supérieure ou égale à un. Lors de la traduction d'une classe de dispositifs de fixation en langage AlbertII, le projet VBA détermine si la cardinalité de la classe générale d'attributs dispositifs de fixation ("1:N") doit faire l'objet d'une contrainte, dite de fixation, rédigée en AlbertII. Celle-ci est en effet vérifiée si la borne inférieure de la cardinalité d'au moins une des classes d'attributs dispositifs de fixation associées à cette classe est supérieure ou égale à un. Dans le cas contraire, la contrainte est exprimée en appliquant un traitement individuel aux attributs dispositifs de fixation. En pratique, nous utilisons les mêmes méthodes pour rédiger les contraintes de composition, de contiguïté, de géométrie et de fixation.

```
%CONSTRUCTED TYPES
//CLASSES DE DISPOSITIFS DE FIXATION
Coller_simple = CP[type:PartFixingFeature,
                   fixation_simple:SET[Attache_pêche_simple]]
     //Contrainte de fixation
     \WITH \ForAll p/Coller_simple :
          Card(fixation_simple) >= 1
Coller_mixte = CP[ type:PartFixingFeature,
                   fixation_mixte_pêche:Attache_pêche_mixte*,
                   fixation_mixte_ananas:Attache_ananas_mixte*,
                   fixation_mixte:SET[Attache_pêche_ananas]]
     //Contrainte de fixation
     \WITH \ForAll p/Coller_mixte :
          Card(fixation_mixte_pêche) = 1 \or
          Card(fixation_mixte_ananas) = 1 \or
          Card(fixation_mixte) >= 1
```

Figure 3.30 : Exemple d'une contrainte AlbertII sur les dispositifs de fixation

La figure 3.30 illustre la contrainte de fixation en reprenant l'exemple de la fabrique de paquets de yogourts décrit à la contrainte précédente. Les pots de yogourts étant collés les uns aux autres, nous ajoutons simplement au modèle la classe de dispositifs de fixation

Coller_simple que nous associons à la classe de dispositifs géométriques *Attache_pêche_simple* par l'intermédiaire de la classe d'attributs dispositifs de fixation *fixation_simple* de cardinalité "0:4", ainsi que la classe de dispositifs de fixation *Coller_mixte* que nous associons aux classes de dispositifs géométriques *Attache_pêche_mixte*, *Attache_ananas_mixte* et *Attache_pêche_ananas* par l'intermédiaire, respectivement, des classes d'attributs dispositifs de fixation *fixation_mixte_pêche* de cardinalité "0:1", *fixation_mixte_ananas* de cardinalité "0:1" et *fixation_mixte* de cardinalité "0:2". Cet exemple ne montre que les contraintes de fixation et les parties de types construits qui y font explicitement référence.

3.1.4.3 Les contraintes invérifiables et intraduisibles

Certaines contraintes liées exclusivement au pattern *Part* ne peuvent être ni vérifiées par le projet de dessin Visio, ni traduites en langage AlbertII.

39. **Une part ne peut être le composant de deux parts composées ou piles**

Le modèle de dessin "Systèmes manufacturiers" est incapable de vérifier la septième contrainte relative aux classes générales car une classe de parts peut être associée à plusieurs classes de parts composées ou de piles par l'intermédiaire de classes d'attributs composants. Il ne peut apporter de solution au problème car, même si nous interdisions à une classe de parts d'être la destination de plusieurs classes d'attributs composants, deux parts composées différentes, instances de la même classe spécifique, pourraient toujours être constituées des mêmes parts composantes. Ce principe induirait en outre une restriction inacceptable parce que, dans un modèle manufacturier, un type de parts ne pourrait servir de composant à plusieurs types de parts composées. Or, des parts aussi courantes que les boulons ou les vis servent à la fabrication d'un très grand nombres d'objets.

Cette contrainte au niveau des instances peut faire l'objet d'une traduction en AlbertII mais pas dans les déclarations des types de données. Nous devons recourir ici à une catégorie particulière de contrainte déclarative qui fait référence aux actions des agents : la *State Behaviour*, une contrainte liée au comportement de l'état d'une part. Or, le pattern *Part* est limité aux déclarations des types de données et aux contraintes qui y sont liées (contraintes d'identité, de composition, de contenance, de cardinalité, d'énumération, de géométrie, de fixation et de contiguïté). La *State Behaviour* impose des contraintes sur l'état des composants au moyen de formules temporelles logiques. Son utilisation doit être limitée au moment où une action spécifiée se produit. La formule temporelle sur l'état des composants doit être vraie chaque fois qu'une occurrence de l'action se déroule mais elle peut être fausse dans les autres cas.

40. **Une part ne change jamais d'identité**

Si une part peut être créée ou détruite et si son état peut évoluer au cours du temps, son identité, par contre, doit toujours rester la même. Nous avons restreint l'identité d'une part aux instances des classes d'identités qui sont associées à la classe de parts dont elle est l'instance. Or, le projet VBA du modèle de dessin "Systèmes manufacturiers" permet seulement d'imposer à chaque part de posséder une identité, tandis que la contrainte d'identité exprimée en langage AlbertII permet uniquement de s'assurer que cette identité conserve un caractère unique. Pour empêcher la modification de l'identité d'une part durant son existence, nous

avons besoin d'un type de contrainte qui puisse recourir à des variables temporelles, ce que ne permettent pas les contraintes liées aux déclarations des types de données.

41. **Un dispositif géométrique assemble les composants directs d'une même part composée**

La quatrième contrainte relative aux classes générales du sous-ensemble du pattern *Part* ne peut pas être vérifiée par la contrainte n° 10 qui exige que toutes les classes de parts composantes associées à une classe de dispositifs géométriques soient associées à la classe de parts composées qui est définie par cette classe de dispositifs géométriques. En effet, si nous reprenons l'exemple de la fabrique de paquets de yogourts décrit aux contraintes n° 37 et 38 dites de géométrie et de fixation, le modèle, qui répond pourtant à toutes les contraintes énumérées précédemment, ne peut empêcher le collage de deux pots de yogourts qui sont les composants de deux paquets différents. Ainsi, nous pouvons assembler des composants de différentes parts composées issues non seulement de la même classe de parts composées comme dans l'exemple cité ici mais également de différentes classes de parts composées grâce au principe de l'héritage dans les hiérarchies de spécialisation (une classe de parts composantes est en effet associée à toutes les sous-classes d'une super classe de parts composées) ou grâce à la possibilité d'associer une classe de parts (composantes) à plusieurs classes de parts composées au travers de relations composants.

La quatrième contrainte sur les classes générales ne peut être traduite en langage AlbertII parce qu'elle nécessite une contrainte déclarative qui fait référence aux actions des agents et qui doit être placée hors des types de données.

42. **Les dispositifs géométriques d'un dispositif de fixation assemblent les composants directs d'une même part composée**

La cinquième contrainte relative aux classes générales n'est pas vérifiée par la contrainte n° 11 qui requiert que toutes les classes de parts composantes associées aux classes de dispositifs géométriques d'une classe de dispositifs de fixation soient associées au travers de relations composants à la classe de parts composées qui est définie par ces classes de dispositifs géométriques, parce qu'elle concerne les classes elles-mêmes et non les instances de ces classes.

D'autre part, si la quatrième contrainte des classes générales n'est pas vérifiable comme nous l'avons démontré ci-dessus, à fortiori, la cinquième ne l'est pas non plus parce que la validité de la première citée est une condition nécessaire à la justesse de la seconde. Celle-ci ne peut pas non plus être traduite en langage AlbertII pour la même raison que celle avancée à la contrainte précédente.

43. **Chaque composant d'une pile est contigu à deux autres composants, à l'exception des deux composants situés aux extrémités qui sont contigus à un seul composant**

La première contrainte relative aux classes spéciales est invérifiable et intraduisible en langage AlbertII car elle ne peut être exprimée qu'au niveau des instances des classes spéciales, c'est-à-dire au niveau des parts composantes, des dispositifs de contiguïté et des attributs dispositifs de contiguïté, soit lors de l'utilisation des modèles graphiques de "parts", à

un moment qui est situé chronologiquement après et donc en dehors du travail de conceptualisation et d'élaboration des modèles graphiques de "parts".

44. **Une pile FIFO, LIFO ou ouverte aux deux extrémités enclot tous ses éléments de telle manière qu'on ne puisse enlever que ceux situés à l'une ou aux deux extrémités**

Nous ne pouvons pas traduire la deuxième contrainte relative aux classes spéciales parce qu'elle s'exerce sur l'opération qui consiste à enlever les parts composant une pile et que cette opération ne peut être traduite que par une action AlbertII. Or, la modélisation du sous-ensemble du pattern *Part* est limitée aux déclarations des types de données et aux contraintes qui y sont liées.

45. **Seules les parts qui sont contenues dans un même conteneur ou qui sont les composants d'une même pile peuvent être contiguës et/ou encloses (entre elles)**

La modélisation graphique des "parts" basée sur le sous-ensemble du pattern *Part* ne permet pas au projet VBA de vérifier la troisième contrainte relative aux classes spéciales parce qu'elle ne peut être exprimée qu'au niveau des instances des classes spéciales. La contrainte n° 19 relative aux classes de dispositifs de contiguïté et la contrainte n° 20 relative aux classes de dispositifs d'enclos vérifient seulement que cette troisième contrainte est réalisable dans le modèle graphique de "parts". Ces deux contraintes ne peuvent empêcher deux parts d'être contiguës ou encloses si elles sont les composants de deux piles différentes, instances de la même classe de piles. Pire encore, puisque les classes d'attributs composants peuvent être héritées par plusieurs classes de piles ou puisqu'une classe de parts peut être le composant de plusieurs autres classes de piles, deux parts issues d'une ou de deux classes de parts différentes et qui sont les composants de plusieurs piles issues elles-mêmes d'une ou de plusieurs classes de piles différentes, peuvent être contiguës ou encloses par plusieurs piles.

Dans l'exemple d'une classe de parts composantes ou contenues qui serait associée à plusieurs classes de piles ou de conteneurs par l'intermédiaire de classes d'attributs composants ou contenants, rien n'interdirait à une part d'être le composant d'une pile ou d'un conteneur et de définir le dispositif de contiguïté d'une autre pile ou d'un autre conteneur, ces piles ou ces conteneurs étant les instances d'une même classe ou de classes différentes. Nous ne pouvons pas non plus vérifier le parallélisme entre les attributs composants et contiguïtés et entre les attributs composants ou contenants et dispositifs d'enclos.

Nous ne pouvons pas non plus traduire cette contrainte en langage AlbertII parce qu'elle est liée aux changements d'états d'une part, soit aux actions AlbertII, et que notre modélisation est limitée aux déclarations des types de données.

46. **Un tampon possède une position fixe**

Le modèle de dessin "Systèmes manufacturiers" permet seulement d'imposer à chaque tampon d'occuper une seule position sans préciser que celle-ci doit posséder un caractère fixe. Pour empêcher la modification de la position d'un tampon durant son existence comme l'exige la quatrième contrainte relative aux classes spéciales, nous avons besoin d'un type de contrainte qui puisse recourir à des variables temporelles, ce que ne permettent pas les contraintes liées aux déclarations des types de données.

47. **Une part ne peut être contenue par deux tampons, ni être à la fois le contenu d'un tampon et le composant d'une part composée ou d'une pile**

Le modèle de dessin "Systèmes manufacturiers" est incapable de vérifier la septième contrainte relative aux classes spéciales car une classe de parts peut être associée à plusieurs classes de parts composées, de piles et/ou de tampons par l'intermédiaire de classes d'attributs composants et/ou contenus. Nous ne pouvons pas nous permettre d'interdire à une classe de parts d'être la destination de plusieurs classes d'attributs composants et/ou contenus parce que deux parts composées, deux piles ou deux tampons, instances de la même classe spécifique, peuvent toujours être constituées de la même part ou contenir la même part, ce qui est interdit, et parce qu'un type de parts doit pouvoir être le composant ou le contenu de plusieurs parts composées, piles et/ou tampons à des moments différents dans un système manufacturier.

À l'instar de la contrainte n° 39 sur les parts composantes, cette contrainte doit être exprimée en langage AlbertII au moyen d'une *State Behaviour*, une contrainte liée au comportement de l'état d'une part et qui fait référence aux actions des agents. Or, le pattern *Part* est limité aux déclarations des types de données et aux contraintes qui y sont liées.

3.1.5. Évaluation du langage de modélisation

L'outil logiciel permet de construire des modèles graphiques de "parts" dans un langage de modélisation particulier. La syntaxe de ce langage est définie par un méta-modèle constitué de méta-classes et de classes spéciales et vérifiée par le modèle de dessin "Systèmes manufacturiers". Quant à sa sémantique, elle est déterminée par le sous-ensemble du pattern *Part* (qui définit notamment des concepts comme les liaisons de spécialisation et d'instanciation, le principe de l'héritage, les attributs, les contraintes, les propriétés et les relations entre les classes) et vérifiée partiellement par le modèle de dessin et plus particulièrement par son projet VBA. Les données relatives à ce langage sont constituées des classes spécifiques instanciées ou spécialisées par l'utilisateur, des propriétés de ces classes (type prédéfini ou non, attribut, cardinalité, duplication, ordre, etc.) et des relations qui associent ces classes.

À l'instar des langages AlbertII, i* et CIMOSA, nous allons examiner la qualité de ce nouveau langage de modélisation pour les systèmes manufacturiers – langage graphique et non plus textuel – à l'aune des trois critères présentés dans la thèse et qui sont la précision (absence d'ambiguïté), le naturel (proximité du langage courant) et l'expressivité (clarté, lisibilité et intelligibilité). [Petit 1999, pp. 25-27] Nous ferons également une comparaison avec le langage AlbertII dans lequel sont traduits les modèles graphiques de "parts" au moyen de l'outil logiciel. Nous verrons ensuite les possibilités de réutilisation introduites par ce nouveau langage et qui constituent une composante essentielle des processus de modélisation des systèmes manufacturiers. [Petit 1999, pp. 27-29]

Le sous-ensemble du pattern *Part* définit un langage graphique, donc considéré a priori comme informel et, par voie de conséquence, peu précis. Ce type de langage souffre en effet d'un très haut degré d'ambiguïté pour les systèmes modélisés car les sémantiques des concepts exprimés par des dessins peuvent être comprises de différentes manières par différentes personnes. Mais notre langage doit-il vraiment être envisagé comme un langage dessiné ? Nous estimons en effet qu'il doit plutôt être considéré comme un langage semi-formel parce qu'il possède un ensemble de concepts, certes limité, mais dont la sémantique intuitive est définie dans la thèse et dans le mémoire, notamment au moyen d'un méta-modèle qui contraint les usages de ces concepts et les relations possibles entre eux. Tous les utilisateurs sont supposés partager cette sémantique et les restrictions imposées par le méta-modèle, réduisant ainsi le risque des interprétations différentes. Cependant, comme la sémantique est définie dans un langage naturel, des malentendus peuvent subsister. D'autre part, cette

sémantique ne permet pas une analyse formelle de la spécification qui nécessite l'apport d'un ensemble de preuves ou de règles d'inférence. Seuls les langages pourvus d'une sémantique exprimée sur la base d'une théorie mathématique ou logique sont reconnus comme formels, ce qui est le cas du langage AlbertII. Ils permettent de faire une analyse de la spécification plus complète que les langages semi-formels (contrôle de cohérence et de type, contrôle du modèle, etc.).

Toutefois, s'il existe une forte relation entre formalisme et précision, celle-ci ne requiert pas nécessairement le formalisme. Un modèle précis peut être exprimé dans un langage naturel. Notre langage possède ainsi une syntaxe qui est définie avec rigueur. La preuve d'une certaine précision est aussi donnée par la possibilité de traduire les modèles graphiques de "parts" en spécifications AlbertII qui sont, elles, sans ambiguïté. Par ailleurs, débuter le processus d'ingénierie par un formalisme strict peut poser des problèmes parce que les besoins ne sont pas toujours précis au départ et sont parfois difficiles à exprimer dans un langage. L'utilisation d'une combinaison d'un langage formel avec un autre moins formel est donc recommandé. Actuellement, les exigences suivent un chemin évolutif d'un statut imprécis et informel (dans la tête du client) vers un statut plus précis qui peut être exprimé d'une manière plus formelle. Par conséquent, la modélisation graphique de "parts" à l'aide de l'outil logiciel permet d'employer un style informel au départ et de faciliter ensuite la transition vers un langage plus formel en traduisant automatiquement les modèles de systèmes en spécifications AlbertII.

Pour que l'activité de modélisation soit efficace, la définition d'un modèle doit être une tâche facile pratiquée d'une manière naturelle. Le caractère naturel d'un langage de modélisation peut être défini à la fois comme la possibilité qui est offerte à ses utilisateurs de représenter aisément et directement la réalité qui doit être modélisée et comme la facilité de compréhension des modèles construits. Cette double qualification requiert des concepts de langage qui possèdent une correspondance directe avec les différentes facettes du système à modéliser. Ces concepts évitent à l'utilisateur d'avoir à encoder certains aspects de la réalité à l'aide d'un langage qui nécessiterait l'utilisation d'un nombre important de concepts de bas niveau et l'écriture d'un nombre élevé de propositions.

Le naturel des langages de modélisation peut être réalisé de deux manières complémentaires : en faisant d'eux des langages spécifiques au domaine d'application et en fournissant des concepts de haut niveau. Dans notre mémoire, le sous-ensemble du pattern *Part* répond au premier critère puisqu'il est en totale adéquation avec le domaine des systèmes manufacturiers pour lequel il a été élaboré. Le pattern *Part* développé dans la thèse contenait déjà des concepts très spécialisés mais nous l'avons encore davantage adapté aux systèmes manufacturiers avec le sous-ensemble. Nous avons ainsi supprimé la classe des parts existantes qui était un concept abstrait pour faire de chaque part une élément réel du système modélisé. Dans le même sens, nous avons simplifié les classes spéciales (tampons, piles et conteneurs) pour une raison de lisibilité des schémas mais également pour qu'à chaque type de parts corresponde une seule classe (et donc aussi une seule forme). Dorénavant, une part spéciale n'est plus représentée par les instances de plusieurs classes spéciales (dont l'une d'entre elles représente l'élément vide de cette classe) mais par une seule classe spéciale (et donc une seule forme dans le modèle de dessin), ce qui correspond mieux à la réalité des systèmes manufacturiers. Le conteneur constitue ici une exception parce qu'il exprime une caractéristique et non un objet réel et parce que nous lui avons octroyé un caractère plus générique que le conteneur décrit dans la thèse. Mais la contradiction n'est qu'apparente parce que nous avons fait du conteneur un concept mieux à même d'exprimer les contraintes d'accessibilité qui s'exercent habituellement sur les parts.

En outre, l'importance que nous accordons à l'aspect pratique de la modélisation nous a incité à adapter les concepts de notre langage à l'application particulière de Visio. Puisque l'instance d'une classe spécifique de parts doit être une part réelle, les classes abstraites ont été supprimées du sous-ensemble (comme les classes de parts existantes) ou ont été assimilées à des calques dans l'outil logiciel (comme les classes de parts et de dispositifs). D'ailleurs, les nouveaux calques ont été créés pour une raison pratique : permettre des recherches sur des groupes de formes dans le code VBA du modèle de dessin.

Notre langage ne satisfait pas au second critère du caractère naturel car il fournit seulement un petit ensemble de concepts de base, obligeant l'utilisateur à écrire de gros modèles pour exprimer des choses simples. Un bon exemple est donné par les dispositifs qui, pour exprimer que deux pièces sont fixées, nécessitent l'utilisation de classes de dispositifs géométriques et de fixation qui doivent être associées avec une classe de parts composées et avec plusieurs classes de parts composantes. C'est également le cas de la caractéristique d'enclosure des éléments d'une pile qui exige à elle seule l'usage de classes de dispositifs de contiguïté et d'enclos qui doivent être associées avec la classe de piles et l'ensemble des classes de parts composantes. En outre, il existe autant de classes de dispositifs géométriques ou de contiguïté que d'associations entre des classes de parts composantes différentes. Dans l'exemple de modélisation graphique de "parts" à l'annexe D, la contiguïté des poids du tracteur (six petits poids au centre et trois gros de chaque côté) nécessite à elle seule trois classes de dispositifs de contiguïté différentes. Mais comment notre langage si spécifique pourrait-il définir des concepts de haut niveau qui seraient une combinaison de plusieurs concepts simples et qui fourniraient une description de plusieurs aspects de la réalité en une seule fois plutôt que d'avoir à décrire tous ces aspects séparément ?

Le langage AlbertII est moins naturel que le nôtre car il est plus général et fournit des concepts moins spécifiques au domaine d'application. Cette remarque est surtout valable pour son aspect formel qui requiert de penser plus rigoureusement et d'écrire des modèles dans une syntaxe contraignante. Cependant, AlbertII a été conçu avec l'objectif d'être naturel et les patterns de contraintes, spécialement ceux des contraintes déclaratives, sont des exemples de concepts qui rendent ce langage plus naturel.

Pour modéliser des systèmes complexes comme les systèmes manufacturiers, un grand nombre d'aspects doivent être pris en compte. Un bon langage de modélisation doit être suffisamment expressif pour fournir une représentation de tous les aspects importants du système considéré. En raison du petit nombre de concepts définis par le sous-ensemble du pattern *Part*, notre langage ne peut évidemment se targuer de pouvoir représenter à lui seul tous les objets physiques destinés à être développés et produits par les systèmes manufacturiers. La réalité est infinie et notre outil logiciel est déjà incapable de modéliser l'assemblage de pièces qui ne sont pas des composantes d'une même part composée. Notre langage est sans doute trop spécifique au domaine que pour pouvoir exprimer l'entièreté d'un système. Toutefois, il a été conçu pour prendre en compte un certain nombre de réalités propres aux systèmes manufacturiers. Cette volonté est à l'origine de la création des classes spéciales qui se situent un peu en marge du méta-modèle de classes mais qui ont largement prouvé leur utilité concrète en permettant de modéliser des aspects particuliers dans la description des parts. Par ailleurs, nous avons conféré au conteneur un caractère plus générique dans notre sous-ensemble afin d'étendre la notion de contraintes d'accessibilité. Nous pourrions faire de même avec d'autres concepts comme les dispositifs géométriques en offrant la possibilité d'assembler des parts qui ne sont pas liées entre elles par des relations de composition.

Les concepts développés dans le sous-ensemble du pattern *Part* trouvent leurs correspondants dans les types de données de la partie déclaration des spécifications AlbertII. Ces types de données correspondent à la dimension information de CIMOSA qui concentre la modélisation de l'information utilisée et produite par les processus d'entreprise. Or, le langage AlbertII est beaucoup plus expressif que notre langage parce qu'il fournit des concepts puissants pour décrire les modèles conceptuels présents dans la dimension information de CIMOSA. Mais notre langage apporte une meilleure lisibilité des parts que les types de données AlbertII parce qu'il permet de visualiser leurs relations (composition, spécialisation), leurs caractéristiques (physiques, géométriques, de contiguïté, d'enclos) et de se concentrer sur certains aspects ou d'éviter les surcharges en découpant le modèle en sous-modèles qui peuvent être groupés par thèmes.

Un bon langage de modélisation doit aussi offrir des concepts qui fournissent ou facilitent la réutilisation de modèles. En effet, afin d'obtenir une gestion efficace du changement, les entreprises doivent gérer leur propre processus de (ré)ingénierie. Or, la modélisation des systèmes manufacturiers représente une part importante de l'effort d'ingénierie et les langages de modélisation sont des composants essentiels utilisés à chaque étape des méthodologies d'ingénierie. Les modèles permettent à des personnes différentes impliquées dans le processus de communiquer, de maîtriser la complexité des systèmes manufacturiers, de comprendre et d'analyser la situation, de pratiquer la (ré)ingénierie du système et, finalement, de contrôler celui-ci. Le processus de réutilisation peut se faire de deux manières : soit par le réemploi de la connaissance acquise durant le processus d'ingénierie antérieur, soit par la réutilisation de la connaissance générique qui ne se rapporte pas directement au système existant mais qui est applicable à ce système.

La réutilisation de la connaissance sur le système consiste à reprendre, d'une part, les modèles existants et, d'autre part, la connaissance sur le processus suivi pour construire ces modèles. Les changements incrémentaux du système existant sont souvent la règle car les changements complets sont rares et hasardeux en raison du risque qu'ils représentent. Le réemploi de modèles anciens consiste à les adapter pour représenter ce changement incrémental sans reconstruire le modèle depuis le début. Dans le cas de notre modélisation graphique de "parts", cela reviendrait à reprendre une classe sans devoir la redéfinir ou à reprendre un morceau du modèle en ne modifiant que la partie concernée par l'incrémentation. Pour l'instant, notre outil logiciel ne le permet pas mais cette réutilisation représente une amélioration possible.

Nous pouvons envisager celle-ci de deux manières. La première consiste à placer dans un nouveau projet (document) les changements opérés, ainsi que les anciennes classes (de parts) non modifiées sans leurs définitions (classes d'attributs et d'états). Dans ce cas, nous devons instaurer une liaison entre les différents projets et, lors de la traduction, récupérer la définition des classes reprises dans ces projets sans traduire l'entièreté de ceux-ci. Nous pouvons aussi simplement redéfinir les classes à modifier dans un nouveau projet et traduire cette fois-ci l'ensemble des projets (le nouveau et les anciens) en écrasant les définitions antérieures par les nouvelles. La seconde manière de réutiliser les modèles existants consiste à stocker ceux-ci dans un repository servant de base à la plate-forme d'outil logiciel et à charger tout ou partie de ces modèles dans un nouveau projet qui modélise l'entièreté du système après incrémentation et qui sera lui aussi sauvegardé dans le repository. Dans la méthode précédente, ce sont les anciens projets qui font office de bases de données.

Comme les changements sont souvent incrémentaux, la plupart des processus suivis pour la construction initiale du système peuvent être révisés avec des modifications locales. Pour être en mesure de refaire ces processus, leurs traces doivent être enregistrées. Idéalement,

elles doivent être consignées explicitement, d'où l'importance de la traçabilité (d'une exigence) qui est définie comme la capacité de décrire et de suivre une exigence à travers l'entièreté de son cycle de vie.

La traçabilité est réalisée globalement si nous sommes capables de déterminer les relations qui existent entre :

- les besoins mis sur le système existant;
- les objets dessinés qui ont été conçus pour rencontrer ces besoins et;
- les objets fabriqués dans le système qui implémentent ces objets dessinés.

La réutilisation de la connaissance sur le processus de modélisation d'un système existant est impossible à réaliser à partir de notre outil logiciel car les informations requises ne sont pas enregistrées : notre langage se limite à modéliser les parts des systèmes manufacturiers sans tenir compte des besoins du système qui sont à l'origine de la modélisation ni des objets réels qui incarnent les instances des classes dessinées. Pour faire la liaison entre les modèles graphiques de "parts", les besoins du système et les pièces employées dans la réalité, un nouveau logiciel externe est nécessaire.

La seconde méthode de réutilisation, soit la réutilisation de la connaissance générique, consiste, d'une part, à reprendre les modèles de systèmes abstraits et, d'autre part, à appliquer des transformations génériques de modèles. À l'opposé du réemploi de la connaissance liée au système existant, cette méthode concerne l'information qui a été abstraite de plusieurs projets de modélisation. Cette information peut concerner des modèles génériques ou des étapes de processus génériques de modélisation.

Les modèles génériques sont ceux construits par les modélisateurs de l'entreprise par abstraction des particularités des applications et par factorisation des aspects communs des fragments de modèles. Ils peuvent être extraits des librairies de ces modèles. Les modèles génériques représentent les combinaisons récurrentes de concepts de modélisation et sont parfois désignés pour un domaine d'application particulier. En outre, les modèles réutilisables groupent souvent des ensembles de concepts relatifs au domaine spécifique dans des structures plus larges. Ils contribuent aussi à faire de la modélisation plus naturelle. Un exemple de modèles réutilisables sont les classes prédéfinies des processus de domaines spécifiques comme les procédures de comptage ou les protocoles de communication. Un langage qui fournit des composants réutilisables devrait permettre de les adapter facilement à un système très proche. Il pourrait aussi être possible d'étendre l'ensemble des modèles réutilisables en ajoutant de nouveaux composants. Comme indiqué ci-dessus, l'outil logiciel ne permet pas actuellement de réutiliser d'anciens modèles mais une telle procédure peut être mise en place soit en établissant une liaison avec des projets antérieurs, soit en construisant une base de données qui intégrerait l'ensemble des anciens modèles et qui permettrait de réemployer tout ou partie de ceux-ci.

L'usage de la connaissance générale sur les processus de modélisation concerne des fragments de processus de modélisation qui sont fréquemment exécutés par les modélisateurs. Ces fragments correspondent à des améliorations et des transformations de modèles effectuées fréquemment ou avec succès et que les modélisateurs peuvent, dans une situation donnée, appliquer en une seule étape sans faire de raisonnements complexes sur leur validité et leur adéquation au contexte. Ils peuvent être définis sur la base des traces de procédures suivies en modélisant les applications particulières comme exprimé ci-dessus. Les morceaux de processus génériques représentent des blocs qui peuvent être utilisés pour construire un modèle explicite d'un processus de modélisation ou d'ingénierie dans le but de permettre d'analyser ce processus. Ils sont aussi employés pour les processus relatifs à l'ingénierie de l'entreprise, des besoins et des bases de données. Comme nous l'avons écrit ci-dessus, notre

outil logiciel est trop limité et trop spécifique pour prendre en considération l'ensemble ou une partie du cycle de vie de la spécification d'une part.

La possibilité d'implémenter la réutilisation de modèles antérieurs au système existant pour modifier celui-ci ou de modèles génériques pour construire un nouveau modèle du système existant constitue une nette amélioration par rapport au langage AlbertII qui ne fournit pas beaucoup d'aide pour définir des modèles réutilisables. Ce dernier ne possède en effet pas de notion de spécialisation sur les types de données et ne supporte pas non plus la paramétrisation. Notre outil logiciel pourrait être couplé à un repository qui permettrait de reprendre des modèles anciens ou des fragments de ceux-ci sous forme de classes prédéfinies (avec leurs classes d'attributs et d'états). Nous pourrions également offrir la possibilité d'instaurer des liaisons de spécialisation entre les classes des nouveaux projets et les classes déjà présentes dans la base de données.

3.2. Définition d'un sous-langage AlbertII pour le sous-ensemble du pattern *Part*

Cette section se situe dans le prolongement de la présentation générale du langage AlbertII faite au premier chapitre. Nous avons déjà mentionné qu'AlbertII était un langage formel orienté agents destiné à la spécification des besoins logiciels. Nous l'avons également décrit comme un langage expressif et naturel. Nous allons nous focaliser ici sur un sous-ensemble de ce langage utilisé pour exprimer les modèles de "parts" et dont nous allons définir les caractéristiques et la syntaxe concrète.

3.2.1. Les caractéristiques du sous-langage AlbertII

Le sous-ensemble du pattern *Part* concerne uniquement la partie déclaration des types de données à laquelle s'ajoute la définition des contraintes qui s'y rapportent. Nous ne nous intéresserons donc ni aux actions, ni aux agents. Quant aux patterns de contraintes, nous les réutiliserons, parfois en les adaptant, pour exprimer les contraintes d'identité, de composition, de contenance, de cardinalité, de contiguïté, d'énumération, de géométrie et de fixation.[41]

La traduction des modèles graphiques de "parts" en langage AlbertII nécessite l'utilisation des types de données abstraites suivants :

- les types de données élémentaires (*BasicType*) prédéfinis : chaîne de caractères (*STRING*), caractère (*CHAR*), booléen (*BOOLEAN*), entier (*INTEGER*), rationnel (*RATIONAL*) et durée (*DURATION*);
- les types élémentaires (*BasicType*) définis par l'utilisateur;
- les types construits (*ConstructedType*) définis par l'utilisateur et qui sont élaborés à l'aide des constructeurs de types suivants : produit cartésien (*CP*), ensemble (*SET*), séquence (*SEQ*), sac (*BAG*) et énumération (*ENUM*).

Seuls les types des deux dernières catégories doivent être déclarés explicitement, les autres étant préétablis dans la spécification.

[41] Pour un exemple de modélisation graphique de "parts" avec sa traduction en AlbertII, voir l'annexe D.

Un type peut recevoir la valeur supplémentaire *UNDEF* par l'adjonction d'une étoile (*) placée à la fin de son nom. Dans ce cas, les valeurs permises pour le type augmenté sont les valeurs du type original plus la valeur *UNDEF*.

Le constructeur d'énumérations (*ENUM*) permet de définir un type de données sur la base d'un autre en déclarant une contrainte (un invariant) qui sélectionne un sous-ensemble des instances du type de base comme les instances du nouveau type. Cette technique est explicitée dans la section suivante par la règle 29 relative à la traduction des super classes de parts racines en langage AlbertII.

Les huit contraintes sont employées dans le sous-ensemble du pattern *Part* afin de restreindre l'ensemble des comportements possibles d'une part (ou d'une classe de parts) en un certain nombre de comportements admissibles. Elles sont exprimées sous la forme de différents modèles de propriétés disponibles dans le langage qui appartiennent exclusivement à la famille des contraintes déclaratives. Celle-ci groupe tous les modèles qui ne sont pas opérationnels. Elle décrit, souvent de manière non déterministe, les états ou actions qui sont distants dans le temps, à l'opposé des contraintes opérationnelles qui décrivent les actions dans les états adjacents.

Les contraintes de composition, de contenance, de cardinalité, de contiguïté, d'énumération, de géométrie et de fixation utilisent l'opération "*Card*" qui est une opération prédéfinie sur les types de données AlbertII. Elle retourne le nombre d'éléments d'un ensemble qui est ici l'ensemble des valeurs permises par les types correspondant aux classes de parts, de dispositifs physiques, de dispositifs géométriques, de dispositifs de contiguïté et de dispositifs de valeur. Définie au départ pour les types conçus au moyen du constructeur d'ensembles (*SET*), nous l'avons redéfinie pour d'autres types car aucune opération similaire à "*Card*" n'existait pour ces types dans le manuel de référence de l'éditeur AlbertII. [Albert 1997]

Soit T un ensemble, voici la syntaxe concrète de l'opération "*Card*" pour les types de données conçus :

- au moyen du constructeur d'ensembles : SET[T] -> INTEGER;
- au moyen du constructeur de sacs : BAG[T] -> INTEGER;
- au moyen du constructeur de séquences : SEQ[T] -> INTEGER;
- sans constructeur : T -> INTEGER.

Card(s) retourne le nombre de membres de s.

3.2.2. La syntaxe concrète du sous-langage AlbertII

La syntaxe concrète du sous-langage AlbertII est présentée à l'annexe C.

La traduction du modèle graphique de "parts" est enregistrée dans un fichier texte portant l'extension ".txt" dont le nom et la localisation peuvent être choisis par l'utilisateur. Le nom et le chemin par défaut sont respectivement "Albert" et "C:\". Le titre du code AlbertII est placé à la première ligne du fichier. Il reprend le nom du document Visio (projet de dessin) contenant le modèle graphique de "parts". Il est précédé du mot-clé "%SPEC". Le reste du fichier contient la traduction des classes spécifiques et se répartit en deux sections.

La première présente les types de données de base et la seconde les types de données construits, chacune étant respectivement précédée des mot-clés "%BASIC TYPES" et "%CONSTRUCTED TYPES". La seconde est elle-même subdivisée en huit parties ordonnées comme suit : les types des classes de parts élémentaires, les types des super classes de parts racines, les types des classes de parts sous-ensembles et super classes, les types des sous-classes de parts feuilles, les types des classes de dispositifs physiques, les types des classes de dispositifs géométriques, les types des classes de dispositifs de fixation, les types des classes de dispositifs de contiguïté et les types des classes de dispositifs d'enclos.

3.3. Règles de mappage du sous-ensemble du pattern *Part* vers le langage AlbertII

Dans la thèse, le pattern *Part* est uniquement traduit en langage CIMOSA parce que, contrairement à AlbertII, il permet d'exprimer la spécialisation des objets d'entreprise. Une traduction en AlbertII peut être obtenue dans un deuxième temps en réutilisant les règles de mappage qui définissent les aspects informationnels des domaines CIMOSA et de leurs fonctionnalités. [Petit 1999, pp. 115-116 et 194-195] Par conséquent, nous ne pouvons pas reprendre à notre compte les règles de mappage existantes : nous devons les réinventer pour traduire directement le sous-ensemble du pattern *Part* en langage AlbertII.[42]

Les nouvelles règles de mappage établissent une liaison des méta-classes et des classes spéciales du sous-ensemble vers les méta-classes AlbertII qui sont le type de données de base (*BasicType*) et le type de données construit (*ConstructedType*), tandis que leur application associe les classes du sous-ensemble créées par l'instanciation des méta-classes et la spécialisation des classes spéciales aux instances des méta-classes AlbertII qui sont les types de données spécifiques de base et les types de données spécifiques construits. Ces règles transposent chaque instance d'une méta-classe (de parts ou d'états) et chaque sous-classe d'une classe spéciale (de parts ou d'états) du sous-ensemble en un type de données spécifique, parfois accompagné de contraintes d'identité, de composition, de contenance, de cardinalité, de contiguïté, d'énumération, de géométrie ou de fixation.

Tous les types de données construits AlbertII sont des produits cartésiens à l'exception de ceux qui traduisent les classes de parts qui sont à la fois des sous-ensembles et des super classes. Ces derniers types se présentent sous la forme d'une équivalence.

Quelle que soit la méthode choisie pour réutiliser le sous-ensemble (copier/coller, paramétrisation, instanciation ou spécialisation – seules les deux dernières sont employées ici), les règles de traduction qui doivent être appliquées sont les mêmes puisque les classes obtenues par réutilisation sont toutes des instances des méta-classes et parfois aussi des sous-classes des classes spéciales du sous-ensemble du pattern *Part*.

Les relations qui existent entre les concepts du langage AlbertII et du sous-ensemble du pattern *Part* représentent explicitement les règles de traduction ou de transformation entre les deux langages dans l'optique d'une approche transformationnelle. Le méta-modèle de chaque langage reste inchangé et des liaisons sont créées entre les éléments des modèles exprimés dans les différents langages en appliquant les règles de mappage. Ces règles sont uniquement employées pour créer des modèles AlbertII à partir de modèles graphiques de "parts" au moyen d'un processus de transformation. L'approche associative qui est à l'opposé

[42] Pour un exemple de modélisation graphique de "parts" avec sa traduction en AlbertII, voir l'annexe D.

établit plutôt des liaisons entre les concepts de différents langages afin d'exprimer la correspondance entre des modèles existants qui ont été créés indépendamment.

Deux schémas à l'annexe E illustrent la définition et l'usage de règles de mappage sur deux fragments de modèles empruntés à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D.

3.3.1. Mappage vers les types de données de base

Règle 1 *Mappe les classes spécifiques créées par instanciation de la méta-classe de dispositifs physiques (*PartPhysicalFeatureClass*) et dont les instances sont des occurrences de dispositifs de part simples en types de données de base définis par l'utilisateur.*

Règle 2 *Mappe les classes spécifiques créées par instanciation des méta-classes d'identités (*PartIdentityClass*), de positions (*PartPositionClass*) et de dispositifs de valeur (*PartFeatureValueClass*), et dont les noms sont définis par l'utilisateur, en types de données de base définis par l'utilisateur.*

Règle 3 *Mappe les classes spécifiques créées par instanciation des méta-classes d'identités (*PartIdentityClass*), de positions (*PartPositionClass*) et de dispositifs de valeur (*PartFeatureValueClass*), et dont les noms sont prédéfinis, en types de données de base prédéfinis.*

L'équivalent d'un type de données de base au sein du sous-ensemble du pattern *Part* est une classe (d'états) qui ne possède pas de classe d'attributs. Une classe spécifique de parts ne sera donc jamais traduite en type de données de base. En effet, si elle ne fait pas partie d'une hiérarchie de spécialisation, elle est automatiquement associée à une classe d'identités. Par contre, si elle est membre d'une telle hiérarchie, soit elle en est la super classe racine et elle dispose d'une classe d'attributs identités, soit elle est une sous-classe et son type est donné par sa super classe directe à laquelle elle est associée par une relation de spécialisation. Une classe de dispositifs géométriques, de fixation, de contiguïté ou d'enclos ne sera jamais non plus exprimée sous la forme d'un type de base car, par définition, elle possède au moins une classe d'attributs dispositifs géométriques, de fixation, de contiguïté ou d'enclos.

3.3.2. Mappage vers les types de données construits

1. **Les classes de parts et d'états**

Règle 4 *Mappe les classes spécifiques créées par instanciation des méta-classes de parts de base (*BasicPartClass*), de parts composées (*CompoundPartClass*) et de tampons (*Buffer*) et par spécialisation des classes spéciales de piles (*PileOfParts*) et de conteneurs (*Container*), et qui ne font pas partie d'une hiérarchie de spécialisation, en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 5 *Mappe les classes spécifiques créées par instanciation des méta-classes de parts mixtes (*BasicOrCompoundPartClass*), de parts composées (*CompoundPartClass*), de parts de base (*BasicPartClass*) et de tampons (*Buffer*) et par spécialisation des classes spéciales de piles (*PileOfParts*) et de conteneurs (*Container*), et qui constituent les racines des*

*hiérarchies de spécialisation, en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 6 *Mappe les classes spécifiques créées par instanciation des méta-classes de parts mixtes (*BasicOrCompoundPartClass*), de parts composées (*CompoundPartClass*), de parts de base (*BasicPartClass*) et de tampons (*Buffer*) et par spécialisation des classes spéciales de piles (*PileOfParts*) et de conteneurs (*Container*), et qui constituent à la fois les sous-classes et les super classes des hiérarchies de spécialisation, en types de données construits : le type de chaque sous-classe devient celui de sa super classe directe (c'est-à-dire de la classe située au niveau supérieur immédiat dans la hiérarchie).*

Règle 7 *Mappe les classes spécifiques créées par instanciation des méta-classes de parts composées (*CompoundPartClass*), de parts de base (*BasicPartClass*) et de tampons (*Buffer*) et par spécialisation des classes spéciales de piles (*PileOfParts*) et de conteneurs (*Container*), et qui constituent les feuilles des hiérarchies de spécialisation, en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 8 *Mappe les classes spécifiques créées par instanciation de la méta-classe de dispositifs physiques (*PartPhysicalFeatureClass*) et dont les instances sont des occurrences de dispositifs de part évalués en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 9 *Mappe les classes spécifiques créées par instanciation de la méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*) en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 10 *Mappe les classes spécifiques créées par instanciation de la méta-classe de dispositifs de fixation (*PartFixingFeatureClass*) en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 11 *Mappe les classes spécifiques créées par spécialisation de la classe spéciale de dispositifs de contiguïté (*PartContiguityFeature*) en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

Règle 12 *Mappe les classes spécifiques créées par spécialisation de la classe spéciale de dispositifs d'enclos (*PartEnclosureFeature*) en types de données construits sous la forme de produits cartésiens (*Cartesian product*).*

2. **Les classes d'attributs**

Règle 13 *Mappe les classes d'attributs spécifiques créées par instanciation des méta-classes d'attributs composants (*possible_components*), contenus (*possible_contents*), identités (*possible_identities*), positions (*possible_positions*), dispositifs physiques (*possible_physical_features*) et géométries (*possible_geometry*) et par spécialisation des classes spéciales d'attributs contenants (*contain*), contiguïtés (*contiguity*) et enclos (*enclosure*), et qui appartiennent aux classes spécifiques de parts de base (*BasicPartClass*), de parts composées (*CompoundPartClass*), de tampons (*Buffer*), de piles (*PileOfParts*) et de conteneurs (*Container*) mappées au moyen de la règle 4, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de parts.*

Règle 14 *Mappe les classes d'attributs spécifiques créées par instanciation des méta-classes d'attributs composants (*possible_components*), contenus (*possible_contents*), identités (*possible _identities*), positions (*possible_positions*), dispositifs physiques (*possible_physical_features*) et géométries (*possible_geometry*) et par spécialisation des classes spéciales d'attributs contenants (*contain*), contiguïtés (*contiguity*) et enclos (*enclosure*), et qui appartiennent aux classes spécifiques de parts de base (*BasicPartClass*), de parts composées (*CompoundPartClass*), de parts mixtes (*BasicOrCompoundPartClass*), de piles (*PileOfParts*), de conteneurs (*Container*) et de tampons (*Buffer*) qui constituent les classes des hiérarchies de spécialisation mappées au moyen des règles 5 à 7, en champs des produits cartésiens (*Cartesian product*) correspondant aux super classes de parts racines respectives mappées au moyen de la règle 5.*

Règle 15 *Mappe les classes d'attributs spécifiques créées par instanciation de la méta-classe d'attributs sous-classes (*isA*) et qui appartiennent aux classes spécifiques de tampons (*Buffer*), de parts composées (*CompoundPartClass*), de piles (*PileOfParts*), de conteneurs (*Container*) et de parts de base (*BasicPartClass*) et qui constituent les classes feuilles des hiérarchies de spécialisation mappées au moyen de la règle 7, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de parts.*

Règle 16 *Mappe les classes d'attributs spécifiques créées par instanciation de la méta-classe d'attributs valeurs (*possible_values*) et qui appartiennent aux classes spécifiques de dispositifs physiques (*PartPhysicalFeatureClass*) mappées au moyen de la règle 8, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de dispositifs physiques.*

Règle 17 *Mappe les classes d'attributs spécifiques créées par instanciation des méta-classes d'attributs dispositifs géométriques* (possible_geometrical_features) *et valeurs (*possible_values*) et qui appartiennent aux classes spécifiques de dispositifs géométriques (*PartGeometrical-FeatureClass*) mappées au moyen de la règle 9, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de dispositifs géométriques.*

Règle 18 *Mappe les classes d'attributs spécifiques créées par instanciation des méta-classes d'attributs dispositifs de fixation* (possible_fixing_features) *et valeurs (*possible_values*) et qui appartiennent aux classes spécifiques de dispositifs de fixation (*PartFixingFeatureClass*) mappées au moyen de la règle 10, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de dispositifs de fixation.*

Règle 19 *Mappe les classes d'attributs spécifiques créées par instanciation de la méta-classe d'attributs valeurs (*possible_values*) et par spécialisation de la classe spéciale d'attributs dispositifs de contiguïté (*contiguity_feature*), et qui appartiennent aux classes spécifiques de dispositifs de contiguïté (*PartContiguityFeature*) mappées au moyen de la règle 11, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de dispositifs de contiguïté.*

Règle 20 *Mappe les classes d'attributs spécifiques créées par instanciation de la méta-classe d'attributs valeurs (*possible_values*) et par spécialisation de la classe spéciale d'attributs dispositifs d'enclos (*enclosure_feature*), et qui appartiennent aux classes spécifiques de dispositifs d'enclos (*PartEnclosureFeature*) mappées au moyen de la règle 12, en champs des produits cartésiens (*Cartesian product*) correspondant à ces classes de dispositifs d'enclos.*

3. **Les cardinalités des classes d'attributs**

Règle 21 *Cette règle concerne les classes d'attributs identités qui sont mappées en champs de produits cartésiens au moyen des règles 13 et 14. Les types correspondant aux classes spécifiques d'identités qui sont la destination de ces classes d'attributs constituent les types des champs et ceux-ci possèdent une valeur unique et définie.*

Règle 22 *Cette règle concerne les classes d'attributs composants, contenus, contenants, positions, dispositifs physiques, géométries, dispositifs géométriques, dispositifs de fixation, dispositifs de contiguïté, dispositifs d'enclos, contiguïtés et enclos qui sont mappées en champs de produits cartésiens au moyen des règles 13, 14 et 17 à 20. Les types correspondant aux classes spécifiques de parts, de positions, de dispositifs physiques, de dispositifs géométriques, de dispositifs de contiguïté et de dispositifs d'enclos qui sont la destination de ces classes d'attributs constituent les types des champs et ceux-ci possèdent soit :*

- *une valeur unique et définie si les bornes supérieure et inférieure de la cardinalité de la classe d'attributs équivalent à un et si la classe spécifique de parts qui en est l'origine n'est pas un sous-ensemble;*
- *une valeur unique définie ou non si la borne supérieure de la cardinalité de la classe d'attributs équivaut à un et si, soit la borne inférieure de la cardinalité de la classe d'attributs équivaut à zéro, soit la classe spécifique de parts qui en est l'origine est un sous-ensemble;*
- *un ensemble de valeurs non ordonnées et uniques qui peut être vide dans les autres cas. Le type du champ est conçu au moyen du constructeur d'ensembles (*SET*).*

Règle 23 *Cette règle concerne les classes d'attributs valeurs qui sont mappées en champs de produits cartésiens au moyen des règles 16 à 20. Les types correspondant aux classes spécifiques de dispositifs de valeur qui sont la destination de ces classes d'attributs constituent les types des champs et ceux-ci possèdent soit :*

- *une valeur unique et définie si les bornes supérieure et inférieure de la cardinalité de la classe d'attributs équivalent à un;*
- *un ensemble de valeurs non ordonnées et uniques. Le type du champ est conçu au moyen du constructeur d'ensembles (*SET*);*
- *un ensemble de valeurs non ordonnées dont certaines peuvent être identiques. Le type du champ est conçu au moyen du constructeur de sacs (*BAG*);*
- *un ensemble de valeurs ordonnées dont certaines peuvent être identiques. Le type du champ est conçu au moyen du constructeur de séquences (*SEQ*).*

  Les trois ensembles de valeurs ne peuvent pas être vides.

4. **Les contraintes**

Règle 24 *Pour chaque type de données construit correspondant à une classe de parts citée dans les règles 4 et 5, une contrainte d'identité (un invariant) s'applique à l'ensemble des types des champs correspondant aux classes spécifiques d'attributs identités. Elle spécifie que deux instances d'un même type de données construit ne peuvent posséder des valeurs identiques pour tous les types des champs évoqués ci-dessus.*

Règle 25 *Pour chaque type de données construit correspondant à une classe de parts composées ou de piles citée dans la règle 4, une contrainte de composition (un invariant) s'applique à l'ensemble des types des champs correspondant aux classes spécifiques d'attributs composants à condition que la somme des bornes inférieures des cardinalités de ces classes ne soit pas supérieure ou égale à deux. Elle spécifie qu'une instance du type de*

données construit doit posséder au moins deux valeurs pour l'ensemble des types des champs évoqués ci-dessus.

Règle 26 *Pour chaque type de données construit correspondant à une classe de conteneurs citée dans la règle 4, une contrainte de contenance (un invariant) s'applique à l'ensemble des types des champs correspondant aux classes spécifiques d'attributs contenants à condition que la somme des bornes inférieures des cardinalités de ces classes ne soit pas supérieure ou égale à deux. Elle spécifie qu'une instance du type de données construit doit posséder au moins deux valeurs pour l'ensemble des types des champs évoqués ci-dessus.*

Règle 27 *Pour chaque type de données construit correspondant à une classe de parts de base, de parts composées, de piles, de conteneurs, de tampons, de dispositifs physiques, de dispositifs géométriques, de dispositifs de fixation, de dispositifs de contiguïté ou de dispositifs d'enclos citée dans les règles 4 et 8 à 12, une contrainte de cardinalité (un invariant) s'applique au type de chaque champ correspondant à une classe spécifique d'attributs composants, contenus, contenants, dispositifs physiques, géométries, dispositifs géométriques, dispositifs de fixation, dispositifs de contiguïté, dispositifs d'enclos, contiguïtés ou valeurs mentionnée dans les règles 22 et 23, à condition que le type de ce champ possède un ensemble de valeurs (c'est-à-dire soit conçu au moyen du constructeur* SET, BAG *ou* SEQ*). La contrainte spécifie que le nombre de valeurs du type de ce champ est compris entre les deux bornes de la cardinalité de la classe d'attributs correspondant à ce champ. Toutefois, elle ne tient pas compte de la borne inférieure si celle-ci équivaut à zéro, ni de la borne supérieure si celle-ci équivaut au nombre indéfini ("N" ou "n").*

Règle 28 *Pour chaque type de données construit correspondant à une classe de piles ou de conteneurs citée dans la règle 4, une contrainte de contiguïté (un invariant) s'applique à l'ensemble des types des champs correspondant aux classes spécifiques d'attributs contiguïtés à condition que la somme des bornes inférieures des cardinalités de ces classes ne soit pas supérieure ou égale à un. Elle spécifie qu'une instance du type de données construit doit posséder au moins une valeur pour l'ensemble des types des champs évoqués ci-dessus.*

Règle 29 *Chaque type de données construit correspondant à une classe de parts citée dans la règle 5 est défini sur la base d'un autre type en déclarant une contrainte d'énumération (un invariant) qui sélectionne un sous-ensemble des instances du type de base comme les instances du nouveau type. Celui-ci est toujours choisi parmi les types correspondant aux sous-classes de parts (part de base, part composée, tampon, pile ou conteneur) feuilles citées dans la règle 7. Un champ du type de données construit possède un type conçu au moyen du constructeur d'énumération (*ENUM*) et qui reprend tous les types correspondant aux sous-classes de parts feuilles. La contrainte spécifie pour chacun de ces types que les champs correspondant aux classes d'attributs composants, contenus, contenants, dispositifs physiques, géométries, contiguïtés et enclos dont dispose ou hérite la classe de parts (correspondant à un type) possèdent des valeurs définies tandis que les autres champs correspondant aux classes d'attributs composants, contenus, contenants, dispositifs physiques, géométries, contiguïtés et enclos possèdent des valeurs indéfinies. Elle spécifie également pour chacun de ces types que les contraintes de composition, de contenance, de cardinalité et de contiguïté citées dans les règles 25 à 28 s'appliquent dans les mêmes conditions aux champs correspondant aux classes spécifiques d'attributs composants, contenus, contenants, dispositifs physiques, géométries et contiguïtés dont dispose ou hérite la classe de parts (correspondant à un type).*

Règle 30 *Pour chaque type de données construit correspondant à une classe de dispositifs géométriques citée dans la règle 9, une contrainte de géométrie (un invariant) s'applique à l'ensemble des types des champs correspondant aux classes spécifiques d'attributs dispositifs géométriques à condition que la somme des bornes inférieures des cardinalités de ces classes ne soit pas supérieure ou égale à deux. Elle spécifie qu'une instance du type de données construit doit posséder au moins deux valeurs pour l'ensemble des types des champs évoqués ci-dessus.*

Règle 31 *Pour chaque type de données construit correspondant à une classe de dispositifs de fixation citée dans la règle 10, une contrainte de fixation (un invariant) s'applique à l'ensemble des types des champs correspondant aux classes spécifiques d'attributs dispositifs de fixation à condition que la somme des bornes inférieures des cardinalités de ces classes ne soit pas supérieure ou égale à un. Elle spécifie qu'une instance du type de données construit doit posséder au moins une valeur pour l'ensemble des types des champs évoqués ci-dessus.*

5. **Les commentaires**

Règle 32 *Chaque commentaire associé à une classe spécifique de parts de base, de parts composées, de parts mixtes, de piles, de conteneurs, de tampons, d'identités, de positions, de dispositifs physiques, de dispositifs géométriques, de dispositifs de fixation, de dispositifs de contiguïté, de dispositifs d'enclos ou de dispositifs de valeur citée dans les règles 1 à 12 ou dont le type correspondant est un type prédéfini, est traduit en un commentaire AlbertII. Ce dernier est précédé d'un double slash ("//") et se termine par un saut de ligne manuel.*

Les règles 4 à 12 respectent l'ordre de présentation des types de données construits dans le fichier contenant la traduction du modèle graphique de "parts" en langage AlbertII. Ces types représentent respectivement les classes de parts élémentaires, les super classes de parts racines, les classes de parts sous-ensembles et super classes, les sous-classes de parts feuilles, les classes de dispositifs physiques, les classes de dispositifs géométriques, les classes de dispositifs de fixation, les classes de dispositifs de contiguïté et les classes de dispositifs d'enclos. Avant l'application d'une règle, un commentaire précise dans le code la catégorie de classes de parts ou d'états qui fait l'objet d'une traduction.

Les classes d'attributs sous-classes ne sont pas toujours traduites directement car elles ne représentent pas une propriété des classes de parts ou d'états mais indiquent au contraire les relations de spécialisation qui existent entre les classes de parts.

Les occurrences d'un type correspondant à une classe spécifique d'identités, de positions ou de dispositifs d'enclos possèdent une valeur unique car la borne supérieure de la cardinalité de cette classe est fixée à un. Les classes d'attributs identités, positions et dispositifs d'enclos ne sont donc pas concernées par la contrainte de cardinalité.

Afin de déterminer le type d'une part (part de base, part composée, pile, tampon ou conteneur) et le type d'un dispositif (dispositif physique, dispositif géométrique, dispositif de fixation, dispositif de contiguïté ou dispositif d'enclos), un champ "type" est associé à chaque classe de parts (feuille ou hors hiérarchie) et de dispositifs.

Les contraintes sur les composants et sur les cardinalités interfèrent parfois. Prenons l'exemple d'un moteur composé d'un à quatre pistons. La contrainte de composition exige que le moteur possède au moins deux pistons, tandis que la contrainte de cardinalité commande

qu'il en détienne minimum un et maximum quatre. En résumé, un moteur doit être constitué de deux à quatre pistons. L'extrême rareté de ces interactions et l'importante complexité de leur simplification nous ont incité à ne pas les prendre en considération. Les cas de figure sont en effet très rares car ils ne peuvent se produire que si la somme des bornes inférieures des cardinalités des classes d'attributs composants d'une classe de parts est égale à un. Si elle équivaut à deux ou plus, la contrainte de composition n'est plus nécessaire. Si elle équivaut à zéro, les deux contraintes ne se rencontrent pas car la contrainte de cardinalité ne prend en compte que les bornes supérieures (puisque toutes les bornes inférieures sont égales à zéro), tandis que notre méthode de traduction en AlbertII ne fait pas la distinction entre les différentes cardinalités (elle exige simplement que la somme des cardinalités soit supérieure ou égale à deux, ce qui revient à ne prendre en compte que les bornes inférieures). Si la somme équivaut à un, la contrainte de composition applique un traitement particulier à chaque cardinalité (borne inférieure). Mais la contrainte de cardinalité n'interfère que pour la cardinalité dont la borne inférieure équivaut à un (et si la borne supérieure est différente de un) car, pour la même raison que précédemment, la contrainte de cardinalité ne prend en compte que les bornes supérieures des autres cardinalités (puisque leurs bornes inférieures sont égales à zéro).[43]

[43] Lorsque la somme équivaut à un, la contrainte de composition traite chaque cardinalité individuellement comme la contrainte de cardinalité. La contrainte de composition se présente sous la forme d'une proposition logique composée de propositions de base unies par l'opérateur de disjonction (ou). La contrainte de cardinalité sur la borne inférieure qui se présente sous la forme d'une proposition logique de base est unie à la contrainte de composition par l'opérateur de conjonction (et). Dans ce cas, résumer les deux contraintes revient à distribuer les deux propositions. Cela complexifie donc la rédaction et la compréhension de ces contraintes.

Conclusion

Les deux premiers chapitres du mémoire nous ont permis de présenter les concepts et les outils nécessaires à l'élaboration d'un langage et d'un logiciel destinés à l'édition de spécifications AlbertII dédiées aux systèmes de production manufacturiers. La première et principale source d'informations provient de la thèse de Michael Petit [Petit 1999] dans le prolongement duquel s'inscrit le mémoire. Elle décrit notamment le langage AlbertII employé pour exprimer les spécifications des besoins formels des systèmes manufacturiers. Elle définit également une modélisation de ces systèmes basée sur les patterns et plus particulièrement sur le pattern *Part*. Le logiciel Microsoft Visio constitue la seconde source d'informations et sert de plate-forme pour l'édition graphique de modèles de "parts" et pour leur traduction en langage AlbertII.

Ces modélisations et leur transposition en spécifications orientées agents (AlbertII) ont été permises grâce à la définition, au troisième chapitre, d'un sous-ensemble du pattern *Part* adapté à l'éditeur Visio et d'un sous-langage AlbertII orienté vers les types de données. La correspondance entre le sous-ensemble et le sous-langage a nécessité l'élaboration d'une série de règles de mappage basées sur l'approche transformationnelle. Celle entre le sous-ensemble et les concepts Visio a permis la construction d'un outil logiciel basé sur la réutilisation de formes génériques par instanciation des méta-classes et par spécialisation des classes spéciales que ces formes représentent. Par ailleurs, des contraintes sont nées du sous-ensemble du pattern *Part* et de son adaptation à notre éditeur graphique. Elles ont imposé la vérification de certaines d'entre elles par le modèle Visio et la traduction d'autres en spécifications AlbertII, à charge pour l'utilisateur de contrôler la conformité de son modèle aux contraintes qui ne sont pas vérifiées ou qui ne sont pas traduites directement.

L'outil logiciel permet d'exprimer des modèles de "parts" dans un langage graphique dont la syntaxe et la sémantique sont définies par le sous-ensemble du pattern *Part* comprenant un méta-modèle constitué de méta-classes et de classes spéciales, des contraintes informelles et des commentaires rédigés en langage courant. La syntaxe et la sémantique sont partiellement vérifiées par le modèle de dessin "Systèmes manufacturiers" et plus particulièrement par son projet VBA. Ce langage de modélisation est de type semi-formel car, malgré une syntaxe précise, la sémantique des concepts est définie dans un langage naturel, donc non dépourvu d'ambiguïtés. Mais si la précision du langage est relative, son caractère naturel est indiscutable car les concepts qu'il développe sont en totale adéquation avec le système modélisé. Toutefois, le langage ne possède que des concepts de bas niveau. Il est moins expressif que le langage AlbertII parce qu'il est trop spécifique au domaine d'application que pour pouvoir prendre en considération l'ensemble des aspects liés aux objets physiques des systèmes manufacturiers. Mais il permet une meilleure lisibilité que ce dernier grâce à l'aspect visuel et au découpage (thématique) des spécifications.

Cette lisibilité constitue l'un des principaux apports du mémoire à la modélisation des spécifications des systèmes manufacturiers. L'outil logiciel que nous avons développé permet de visualiser des classes de parts, les relations qui existent entre elles et les classes d'attributs qu'elles possèdent. Dans le cas d'une hiérarchie de spécialisation, par exemple, l'utilisateur visualise directement quelles classes sont associées aux sous-classes et aux super classes de

parts tandis que dans les types de données AlbertII, l'ensemble des classes d'attributs sont regroupées dans la définition de la super classe racine et l'utilisateur doit examiner la contrainte d'énumération pour déterminer les liaisons entre les classes. L'outil logiciel permet aussi d'éclater le modèle en de nombreux sous-modèles et offre en corollaire la possibilité de représenter une même classe à plusieurs endroits du projet. Ces différentes fonctionnalités favorisent la compréhension des définitions de parts (visualisation et découpage), facilitent la recherche d'informations (groupements thématiques et redondants) et rendent le modèle plus attrayant que le code aride des spécifications AlbertII. L'élaboration des modèles est également plus aisée parce que l'utilisateur perçoit mieux le travail de modélisation qu'il a déjà accompli à un moment donné et parce qu'il peut se concentrer sur certains aspects de cette modélisation.

D'autre part, l'outil logiciel facilite le travail du modélisateur en générant automatiquement du code AlbertII dans une syntaxique correcte. Il vérifie aussi partiellement la sémantique (contraintes) et permet une première correction au niveau des modèles graphiques de "parts" en indiquant à l'utilisateur la nature des erreurs et les classes concernées. D'autres contraintes sont directement traduites en contraintes AlbertII. Cette méthode permet d'employer d'abord un style plus informel pour modéliser les exigences du système avant de les transposer vers un langage plus formel.

Le second grand apport du mémoire est la possibilité – non encore implémentée – de réutiliser des modèles antérieurs au système existant pour modifier celui-ci ou de réutiliser des modèles génériques pour construire un nouveau modèle du système existant. Cette possibilité rend les types de données de la partie déclaration des spécifications AlbertII réutilisables alors qu'ils ne l'étaient pas.

Nous avons modifié le pattern *Part* défini dans la thèse en vue de l'améliorer et de l'adapter à l'outil logiciel. Nous avons d'abord corrigé et complété quelques concepts : nous avons rectifié les petites erreurs qui s'étaient glissées dans la description de la classe des piles, nous avons remanié et complété la définition de cette classe pour qu'elle puisse exprimer toutes les subtilités relatives à ce type de parts composées spéciales, nous avons présenté une description graphique de la classe des conteneurs conforme à sa définition dans la thèse, et nous avons modifié les cardinalités des classes générales d'attributs identités et positions afin que l'identité et la position d'une part puissent être représentées par les instances de plusieurs classes.

Nous avons ensuite simplifié le pattern *Part* : nous avons supprimé la méta-classe et la classe générale des parts existantes en déclarant que les systèmes manufacturiers n'emploient que des parts existantes, nous avons modifié la cardinalité de la méta-classe et de la classe générale d'attributs identités en conséquence (toutes les parts doivent dorénavant posséder une identité), nous avons défini la sémantique des classes dans une hiérarchie de spécialisation comme une partition, nous avons banni le principe de l'héritage multiple trop complexe, et nous avons supprimé quelques méta-classes d'attributs, de dispositifs et de parts spéciales, ainsi que les classes générales correspondantes. Ces simplifications n'ont pas restreint la capacité de modélisation du langage car les spécifications associées aux concepts limités ou supprimés peuvent toujours être exprimées au moyen des concepts qui ont été conservés.

Nous avons doté le sous-ensemble du pattern *Part* de plusieurs améliorations : nous avons créé une méta-classe et une classe générale de parts mixtes afin de pouvoir mélanger des classes de parts de base et de parts composées au sein d'une même hiérarchie de spécialisation, nous avons attribué des noms plus explicites à certaines méta-classes et classes générales, nous avons élargi la contrainte d'identité en établissant que deux parts, instances d'une même classe, peuvent posséder une identité différente si elles sont définies par deux instances différentes d'une seule de leurs classes d'identités, et nous avons généralisé les contraintes d'accessibilité en étendant la définition de la classe spéciale des conteneurs.

Nous avons fait preuve de beaucoup de pragmatisme dans l'élaboration de la plate-forme Visio et du langage de modélisation adapté à cette plate-forme parce que l'objectif du mémoire était de concevoir un outil utilisable. Nous avons donc parfois privilégié le côté pratique sans craindre de faire quelques entorses aux principes conceptuels du sous-ensemble du pattern *Part* (notamment sur les anciennes classes spéciales de tampons, de piles et de conteneurs) afin de rendre les modèles plus simples, plus accessibles, plus compréhensibles et plus facilement constructibles. Nous avons tenté de combiner une facilité d'utilisation à une conceptualisation cohérente.

Cependant, la manipulation de l'outil logiciel n'est pas évidente parce que les schémas prennent beaucoup de place et l'espace d'une page se révèle souvent insuffisant pour présenter les définitions des parts complexes. Elle est aussi fastidieuse parce que l'utilisateur doit glisser chaque forme sur le dessin, agrandir les flèches pour relier les classes entre elles et déplacer à tout moment les figures pour éviter les surcharges. À la longue, il risque de se lasser à faire des petits dessins. Par ailleurs, le message d'avertissement qui apparaît lorsqu'une classe est représentée plusieurs fois dans le projet énerve rapidement, surtout si l'on fait de nombreux copier/coller de morceaux de schémas.

L'outil logiciel présente aussi des limitations. Il rend la modélisation extrêmement rigide : seules les formes de base présentes dans les deux gabarits peuvent être employées et elles ne peuvent être modifiées sous peine de provoquer des incohérences avec le code du projet VBA. Ce dernier ne peut pas non plus être révisé, complété ou limité. En conséquence, la génération de spécifications AlbertII n'est réalisable que lorsque le modèle est complet (et cohérent), c'est-à-dire lorsqu'une partie autonome du système existant est entièrement modélisée. D'autre part, l'ensemble du modèle doit être compris dans un seul et même projet (document), ce qui peut se révéler très lourd en terme de pages et, partant, en lisibilité et en manipulation.

L'impossibilité de modifier les formes de base et le projet VBA limite également le langage de modélisation car il empêche l'emploi de deux des trois règles de réutilisation du pattern *Part* : le copier/coller des classes générales et la substitution de paramètre. La rigidité de l'outil logiciel interdit toute personnalisation mais aussi toute évolution des concepts du langage pour les adapter aux situations particulières du système : l'utilisateur doit se contenter de ce que l'outil et le langage lui offrent. Or justement, le langage manque d'expressivité parce qu'il est loin de pouvoir représenter toute la complexité des systèmes manufacturiers.

La portée du langage est aussi restreinte parce qu'elle n'intéresse que les types de données AlbertII qui représentent une faible partie de la modélisation complète d'un système. Et cette spécificité ne permet pas de consigner la trace du processus de modélisation : il est impossible d'établir de liaisons entre les modèles graphiques de "parts" construits, les besoins du système existant et les pièces employées dans le monde réel.

En outre, certaines caractéristiques liées aux parts ne sont pas modélisables au moyen de l'outil logiciel. Ainsi, dans l'exemple de modélisation présenté à l'annexe D, rien ne permet d'indiquer que le plus petit modèle de tracteur possède un petit réservoir à mazout et les deux autres un grand. En effet, la classe de dispositifs physiques indiquant la capacité du réservoir est liée à une classe de parts unique représentant le réservoir des trois modèles. La différence de capacité ne peut être indiquée qu'en commentaire et doit faire l'objet d'une contrainte placée a posteriori hors des types de données AlbertII. Mais cela dépend aussi parfois du choix de modélisation posé par l'utilisateur car, dans notre exemple, nous aurions pu ajouter deux sous-classes à la classe de parts existante ou représenter le réservoir par deux classes de parts différentes attachées chacune à leur(s) modèle(s) respectif(s). De plus, comme nous l'avons vu, une partie des contraintes ne sont pas vérifiables par le modèle de dessin ni traduisibles en AlbertII, notamment celles liées au comportement de l'état d'une part et qui

font référence aux actions des agents. Par conséquent, le code AlbertII généré présente des lacunes et des incohérences, obligeant l'utilisateur à le vérifier, le corriger et le compléter. Ne perdons pas non plus de vue que la cohérence du modèle à l'égard du système représenté repose principalement sur son concepteur.

Un long travail de conceptualisation reste donc à effectuer pour compléter et enrichir les concepts du langage. Il devra poursuivre un double objectif : celui de permettre à l'utilisateur de mieux représenter la réalité des systèmes manufacturiers dans toute leur diversité et celui d'étendre la portée du langage au-delà des types de données AlbertII afin de pouvoir modéliser toutes les caractéristiques liées aux parts et de pouvoir vérifier ou traduire les contraintes qui ne peuvent pas encore l'être. Une étape vers le premier objectif, à savoir l'amélioration de l'expressivité du langage, serait de travailler le caractère générique de certains concepts comme nous l'avons fait avec la classe spéciale de conteneurs. Nous pourrions, par exemple, généraliser les dispositifs géométriques en permettant d'assembler des parts qui ne sont pas liées entre elles par des relations de composition. Nous pourrions également introduire le double héritage ou approfondir la structure des classes d'identités, de positions et de dispositifs de valeur dont le type est défini par l'utilisateur en lui permettant de modéliser les éventuels sous-types de ces classes. Nous pourrions aussi créer de nouvelles classes afin de prendre en considération d'autres aspects communs aux systèmes manufacturiers.

Mais l'extension des concepts à ceux correspondant aux groupements des agents dans les sociétés, aux structures des états et aux actions des agents dans le langage AlbertII est-elle judicieuse ? L'aspect visuel apporte un plus dans la définition d'une part parce qu'elle est un objet de la vie réelle avec des caractéristiques tangibles, ce qui n'est pas le cas des actions ou des états. Comment modéliser ces aspects de manière graphique alors qu'ils présentent moins de prédisposition au dessin ? Cela alourdirait en outre le projet et engendrerait une multiplication des formes de base dans les gabarits au risque de noyer l'utilisateur.

Une autre direction pour l'avenir serait de continuer à simplifier l'outil logiciel et le langage de modélisation. Nous pourrions diminuer certaines contraintes en permettant à une classe (forme) d'être représentée plusieurs fois sur la même page ou en permettant aux piles et aux conteneurs d'enclore des parts sans que celles-ci soient contiguës. Les contraintes invérifiables pourraient faire l'objet d'un commentaire automatique lors de la traduction du modèle graphique de "parts" en langage AlbertII. Nous pourrions également rationaliser la visualisation des modèles en intégrant les classes de dispositifs de valeur à l'intérieur des formes des autres dispositifs et en intégrant certaines classes d'états qui s'y prêtent (identités et positions) à l'intérieur des formes représentant les classes de parts. Mais ces classes pourraient aussi être placées dans les propriétés personnalisées des classes de dispositifs et de parts sans pour autant être affichées à l'intérieur des formes.

La réutilisation des modèles de "parts" reste aussi à implémenter. Elle peut se présenter sous la forme d'une liaison entre différents projets permettant de récupérer la définition des classes comprises dans ces projets ou sous la forme d'un stockage des modèles existants dans un repository servant de base à la plate-forme d'outil logiciel. Visio possède déjà des fonctionnalités permettant de lier des formes à des bases de données à l'aide d'applications ODBC, de générer de nouvelles formes à partir d'enregistrements de bases de données et de créer des dessins qui fonctionnent comme des représentations visuelles de ces données.

En conclusion, le langage de modélisation et l'outil logiciel présentent des lacunes, des limitations et ne satisfont pas à la modélisation des systèmes manufacturiers dans leur ensemble. Mais ils apportent des améliorations notables en terme de lisibilité et de réutilisation. Outre l'implémentation d'un repository, des progrès restent encore à faire pour enrichir les concepts du langage et simplifier l'utilisation de son éditeur graphique.

Bibliographie

[Albert 1997] Manuel de référence pour le langage Albert associé à l'éditeur AlbertII. Rapport de recherche RR-97-002, août 1997. Publication du groupe Albert du Département de Science informatique de l'Université Notre-Dame de la Paix à Namur. http://www.info.fundp.ac.be/cgi-publi/pub-spec-report?RR-97-002.

[Base 2003] Base de connaissances Microsoft – 461121. VBA : Exécuter une macro en passant ses arguments. Rubrique "Aide et support" du site officiel de Microsoft France. http://support.microsoft.com/default.aspx?scid=kb;fr;461121.

[Delannoy 2003] DELANNOY (Claude), *Programmer en Java. Deuxième édition mise à jour*, 2e éd., Eyrolles, Paris, 2003.

[Dictionnaire 1987-2004] Le grand dictionnaire terminologie sur le site officiel de l'Office québécois de la langue française. http://www.granddictionnaire.com/.

[Petit 1999] PETIT (Michaël), *Formal Requirements Engineering of Manufacturing Systems : A Multi-Formalism and Component-Based Approach*, thèse de doctorat, FUNDP, 1999.

[RFC 2234] David Crocker et Paul Overell, "Augmented BNF for Syntax Specifications : ABNF", RFC 2234, novembre 1997. http://www.ietf.org.

[Sciences 2002] Site exposant les concepts fondamentaux de la physique-mathématique et dont le contenu est intégralement soumis à la GNU Free Documentation License. http://www.sciences.ch/htmlfr/geometrie/geometriegraphes01.php.

[Visio 2002] Microsoft Visio Professional 2002.

[Visio 2003] Site officiel de Microsoft Office Visio en France. http://www.microsoft.com/France/office/visio/.

Annexe A

Méta-modèle du sous-ensemble du pattern *Part*

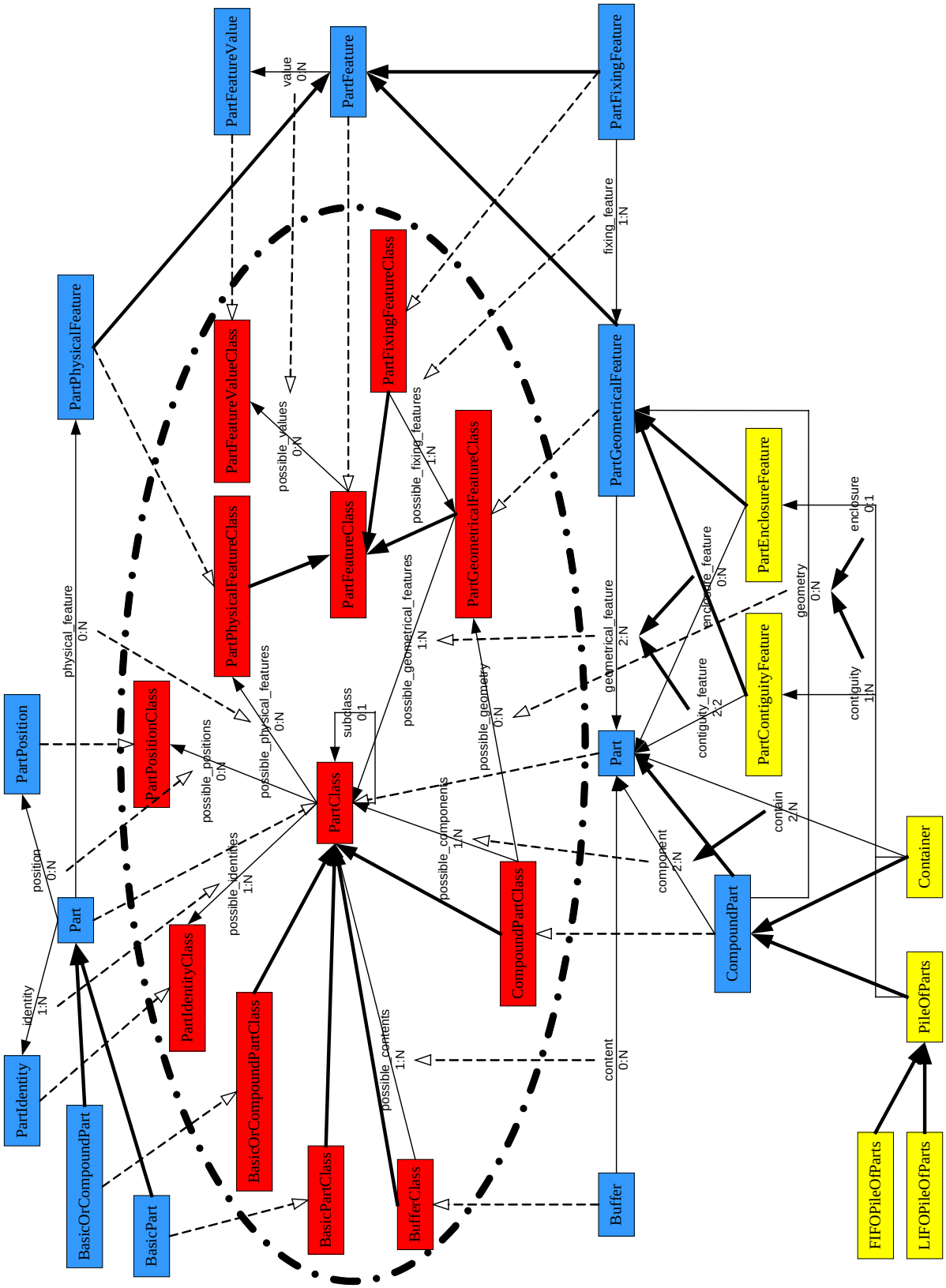


Figure A.1 : Méta-modèle du sous-ensemble du pattern *Part*

Annexe B

Formes de base du modèle "Systèmes manufacturiers"

Les formes de base du modèle de dessin "Systèmes manufacturiers" sont classées selon les calques et accompagnées des informations contenues dans leurs propriétés personnalisées. Les contraintes de cardinalité sur les d'attributs sont exprimées dans les formes par les cardinalités minimales et maximales. Le bloc de texte de chaque forme reprend le nom de la classe par défaut sauf si ce nom peut être emprunté aux types de données élémentaires prédéfinis en langage AlbertII.

1. **La méta-classe de parts (*PartClass*)**

- La méta-classe de parts mixtes (*BasicOrCompoundPartClass*)

Propriétés personnalisées :
1. Nom de la classe

- La méta-classe de parts de base (*BasicPartClass*)

Propriétés personnalisées :
1. Nom de la classe

- La méta-classe de parts composées (*CompoundPartClass*)

Propriétés personnalisées :
1. Nom de la classe

- La classe spéciale de piles de parts (*PileOfParts*)

Propriétés personnalisées[44] :
1. Nom de la classe
2. Type de piles

- La classe spéciale de conteneurs (*Container*)

Propriétés personnalisées :
1. Nom de la classe

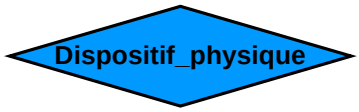
[44] Si le type de piles est FIFO ou LIFO, il apparaît entre parenthèses dans le bloc de texte de la forme.

- La méta-classe de tampons[45] (*Buffer*)

Propriétés personnalisées :
1. Nom de la classe

2. **La méta-classe de dispositifs (*PartFeatureClass*)**

- La méta-classe de dispositifs physiques (*PartPhysicalFeatureClass*)



Propriétés personnalisées :
1. Nom de la classe

- La méta-classe de dispositifs géométriques (*PartGeometricalFeatureClass*)

Propriétés personnalisées :
1. Nom de la classe

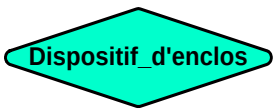
- La méta-classe de dispositifs de fixation (*PartFixingFeatureClass*)

Propriétés personnalisées :
1. Nom de la classe

- La classe spéciale de dispositifs de contiguïté (*PartContiguityFeature*)

Propriétés personnalisées :
1. Nom de la classe

- La classe spéciale de dispositifs d'enclos (*PartEnclosureFeature*)



Propriétés personnalisées :
1. Nom de la classe

[45] Pour marquer la différence entre la classe de tampons et les autres classes de parts dont les instances sont considérées comme de véritables parts, la forme qui représente la classe de tampons est de couleur brune. D'autre part, contrairement aux classes de parts composées, le nom de la classe de tampons n'est pas écrit en petites majuscules et la forme qui la représente n'est pas entourée par un trait épais.

3. **La méta-classe d'états**[46] **(*PartStateClass*)**

- La méta-classe d'identités (*PartIdentityClass*)

Propriétés personnalisées :
1. Nom/Type de la classe

- La méta-classe de positions (*PartPositionClass*)

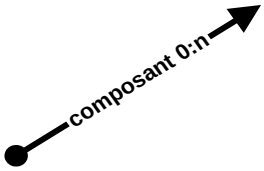
Propriétés personnalisées :
1. Nom/Type de la classe

- La méta-classe de dispositifs de valeur (*PartFeatureValueClass*)

Propriétés personnalisées :
1. Nom/Type de la classe

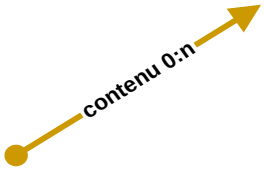
4. **La méta-classe d'attributs**[47] **(*AttributeClass*)**

- La méta-classe d'attributs composants (*possible_components*)



Propriétés personnalisées :
1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La méta-classe d'attributs contenus (*possible_contents*)



Propriétés personnalisées :
1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

[46] La méta-classe d'états n'existe pas dans le sous-ensemble du pattern *Part* : elle est uniquement présente dans le modèle de dessin "Systèmes manufacturiers" car un calque doit être assigné à chaque forme de base comme expliqué à la section 3.1.2.

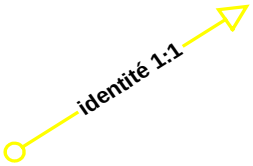
[47] La méta-classe d'attributs n'existe pas dans le sous-ensemble du pattern *Part* : elle est uniquement présente dans le modèle de dessin "Systèmes manufacturiers" pour les raisons pratiques évoquées à la section 3.1.2. Les flèches qui représentent les classes d'attributs reliant les classes de parts entre elles se distinguent par leur trait plein et plus épais.

- La classe spéciale d'attributs contenants (*contain*)

Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La méta-classe d'attributs identités (*possible_identities*)



Propriétés personnalisées[48] :

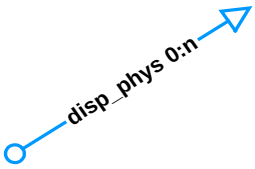
1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La méta-classe d'attributs positions (*possible_positions*)

Propriétés personnalisées[49] :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

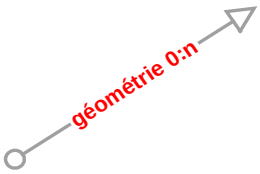
- La méta-classe d'attributs dispositifs physiques (*possible_physical_features*)



Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La méta-classe d'attributs géométries (*possible_geometry*)



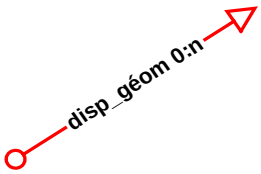
Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

[48] Les deux propriétés personnalisées liées aux bornes sont masquées car la cardinalité obligatoire est : "1-1".

[49] Les deux propriétés personnalisées liées aux bornes sont masquées. À leur place, l'utilisateur doit choisir si l'état position des parts est facultatif (cardinalité : "0-1") ou obligatoire (cardinalité : "1-1"). Cette quatrième propriété personnalisée détermine la valeur de la deuxième propriété personnalisée (borne inférieure de la cardinalité), la valeur de la troisième propriété personnalisée (borne supérieure) étant fixée à "1".

- La méta-classe d'attributs dispositifs géométriques (*possible_geometrical_features*)



Propriétés personnalisées :

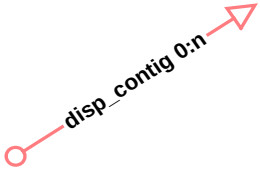
1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La méta-classe d'attributs dispositifs de fixation (*possible_fixing_features*)

Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La classe spéciale d'attributs dispositifs de contiguïté (*contiguity_feature*)



Propriétés personnalisées :

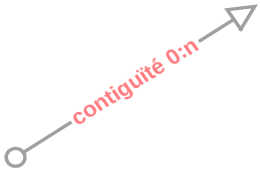
1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La classe spéciale d'attributs dispositifs d'enclos (*enclosure_feature*)

Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

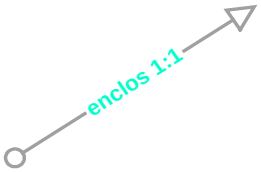
- La classe spéciale d'attributs contiguïtés (*contiguity*)



Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

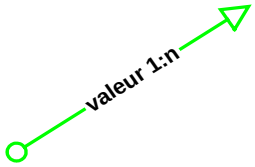
- La classe spéciale d'attributs enclos (*enclosure*)



Propriétés personnalisées[50] :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité

- La méta-classe d'attributs valeurs (*possible_values*)

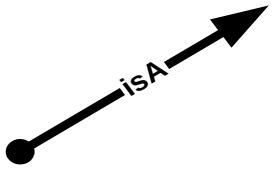


Propriétés personnalisées :

1. Nom de la classe
2. Borne inférieure de la cardinalité
3. Borne supérieure de la cardinalité
4. Duplication des valeurs
5. Ordre des valeurs

5. **La méta-classe de sous-classes**[51] **(*SubClass*)**

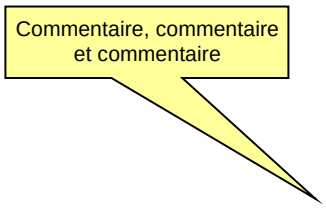
- La méta-classe d'attributs sous-classes (*subclass*)



Propriétés personnalisées : /

6. **La méta-classe de commentaires**[52] **(*CommentClass*)**

- Le commentaire (*Comment*)



Propriétés personnalisées : /
Bloc de texte : chaîne de caractères

[50] Les deux propriétés personnalisées liées aux bornes sont masquées car la cardinalité obligatoire est : "1-1".
[51] La méta-classe de sous-classes n'existe pas dans le sous-ensemble du pattern *Part* : elle est uniquement présente dans le modèle de dessin "Systèmes manufacturiers" pour les raisons pratiques évoquées à la section 3.1.2.
[52] La méta-classe de commentaires n'existe pas dans le sous-ensemble du pattern *Part* : elle est uniquement présente dans le modèle de dessin "Systèmes manufacturiers" pour les raisons pratiques évoquées à la section 3.1.2.

Annexe C

Syntaxe concrète du sous-langage AlbertII

C.1. Présentation de la syntaxe ABNF

La syntaxe concrète partielle du langage AlbertII est décrite au moyen de la méta-grammaire ABNF (Augmented Backus-Naur Form) présentée dans le RFC 2234. [RFC 2234] Elle est basée sur la syntaxe du langage Albert réalisée pour l'éditeur AlbertII. [Albert 1997] Ce manuel de référence donne la sémantique intuitive de tous les constructeurs du langage AlbertII. Il inclut aussi les syntaxes abstraites et concrètes.

Le caractère pour cent (*%*) est utilisé pour débuter une section. Il est suivi par le nom de cette section. Les trois sections que nous employons dans la déclaration des types de données sont ordonnées comme suit : *SPEC* (titre), *BASIC TYPES* (type de base) et *CONSTRUCTED TYPES* (type construit). Cet ordre doit être respecté dans toutes les spécifications. Dans notre configuration, la deuxième section est optionnelle.

Un commentaire qui n'est pas une ligne blanche doit être précédé d'un double slash ("*//*"). Le reste de la ligne est toujours considéré comme un commentaire. Une ligne contient soit du code exécutable, soit un commentaire mais jamais les deux en même temps.

Le mot-clé "*UNDEF*" ne peut jamais être employé comme identifiant pour les types. Les autres mots-clés peuvent être utilisés comme noms pour ces types.

Symboles ABNF	Significations
::=	définit un symbole (terminal) à partir d'autres symboles (terminaux et/ou non terminaux)
(a)	groupement (a est la seule possibilité)
a | b	dénote l'alternative entre a et b
[a]	a est optionnel (facultatif)
{a}*	représente aucune, une ou plusieurs occurrences de a
$\{a\}^+$	représente au moins une occurrence de a
"a"	a représente un terminal (a est insécable)
<a>	a représente un non terminal
{a-b}	représente un élément parmi l'intervalle de a à b

Tableau C.1 : Légende de la notation en ABNF

Le tableau C.1 présente les conventions d'écriture de la notation en ABNF. Les opérateurs sont classés selon leur ordre de précédence. Les lettres "a" et "b" représentent chacune une liste d'objets. Dans le contexte du sous-ensemble du pattern *Part*, le plus petit objet est un caractère. Les terminaux sont des séquences de caractères invariables et indéfinissables : ils apparaissent sous cette forme dans le code AlbertII. Ils sont écrits en gras et en italique dans la syntaxe. Les non terminaux sont toujours définis par un ensemble de terminaux et/ou de non terminaux.

Pour améliorer la lisibilité de la syntaxe, nous avons décidé de ne pas toujours représenter l'espace blanc précédent ou suivant un terminal par son symbole (non terminal) mais plutôt de les fusionner afin d'obtenir un seul terminal. Dans les autres cas, nous emploierons le symbole non terminal.

La syntaxe concrète du sous-langage AlbertII présentée ci-dessous colle au plus près de la traduction des modèles graphiques de "parts" par le logiciel Visio : sa structure est strictement adaptée au code généré. Elle respecte notamment l'ordre de traduction des catégories de classes et de contraintes.

C.2. La syntaxe ABNF du sous-langage AlbertII

<SPEC> ::=
    <TITLE-DECL> <DT-DECLS>

<TITLE-DECL> ::=
    <HDR-TITLE> <SPACE> <STRING> <NL> <NL>
<DT-DECLS> ::=
    [<HDR-BASIC-TYPES> {<BASIC-DT-DECL>}$^{+}$]
    <HDR-CONSTR-TYPES> <CONST-DT-DECL>

<BASIC-DT-DECL> ::=
    <IDENT> <NL>
    {<SPACE4> <COMMENT>}*
    <NL>
<CONST-DT-DECL> ::=
    <CONST-ELEMENTARY-DECL>
        [<CONST-PHYSICAL-DECL>]
        [<CONST-GEOMETRICAL-DECL> [<CONST-FIXING-DECL>]]
        [<CONST-CONTIGUITY-DECL> [<CONST-ENCLOSURE-DECL>]]
    | <CONST-ROOT-DECL>
        [<CONST-SUB-CLASS-DECL>] <CONST-LEAF-DECL>
        [<CONST-PHYSICAL-DECL>]
        [<CONST-GEOMETRICAL-DECL> [<CONST-FIXING-DECL>]]
        [<CONST-CONTIGUITY-DECL> [<CONST-ENCLOSURE-DECL>]]
    | <CONST-ELEMENTARY-DECL>
        <CONST-ROOT-DECL>
        [<CONST-SUB-CLASS-DECL>] <CONST-LEAF-DECL>
        [<CONST-PHYSICAL-DECL>]
        [<CONST-GEOMETRICAL-DECL> [<CONST-FIXING-DECL>]]
        [<CONST-CONTIGUITY-DECL> [<CONST-ENCLOSURE-DECL>]]

<CONST-ELEMENTARY-DECL> ::=
<CONST-ELEMENTARY-TITLE>
{<TYPE-ELEMENTARY>}+
<CONST-ROOT-DECL> ::=
<CONST-ROOT-TITLE>
{<TYPE-ROOT>}+
<CONST-SUB-CLASS-DECL> ::=
<CONST-SUB-CLASS-TITLE>
{<TYPE-SUB-CLASS>}+
<CONST-LEAF-DECL> ::=
<CONST-LEAF-TITLE>
{<TYPE-LEAF>}+
<CONST-PHYSICAL-DECL> ::=
<CONST-PHYSICAL-TITLE>
{<TYPE-PHYSICAL>}+
<CONST-GEOMETRICAL-DECL> ::=
<CONST-GEOMETRICAL-TITLE>
{<TYPE-GEOMETRICAL>}+
<CONST-FIXING-DECL> ::=
<CONST-FIXING-TITLE>
{<TYPE-FIXING>}+
<CONST-CONTIGUITY-DECL> ::=
<CONST-CONTIGUITY-TITLE>
{<TYPE-PHYSICAL>}+
<CONST-ENCLOSURE-DECL> ::=
<CONST-ENCLOSURE-TITLE>
{<TYPE-PHYSICAL>}+

<TYPE-ELEMENTARY> ::=
<IDENT> " = " "***CP[***" <TYPE-LIST-CP-ELEMENTARY> "***]***" <NL>
{<TAB> <COMMENT>}*
<CONSTRAINT-IDENTITY>
[<CONSTRAINT-COMPONENT> | <CONSTRAINT-CONTAIN>]
[<CONSTRAINT-CARDINALITY>]
[<CONSTRAINT-CONTIGUITY>]
{<SPACE4> <COMMENT>}*
<NL>
<TYPE-ROOT> ::=
<IDENT> " = " "***CP[***" <TYPE-LIST-CP-ROOT> "***]***" <NL>
{<TAB> <COMMENT>}*
<CONSTRAINT-IDENTITY>
<CONSTRAINT-ENUMERATION>
{<SPACE4> <COMMENT>}*
<NL>
<TYPE-SUB-CLASS> ::=
<IDENT> " = " <IDENT> <NL>
{<SPACE4> <COMMENT>}*
<NL>

<TYPE-LEAF> ::=
<IDENT> " = " "***CP[***" <TYPE-LIST-CP-LEAF> "***]***" <NL>
{<SPACE4> <COMMENT>}*
<NL>
<TYPE-PHYSICAL> ::=
<IDENT> " = " "***CP[***" <TYPE-LIST-CP-FEATURE> "***]***" <NL>
{<TAB> <COMMENT>}*
[<CONSTRAINT-CARDINALITY>]
{<SPACE4> <COMMENT>}*
<NL>
<TYPE-GEOMETRICAL> ::=
<IDENT> " = " "***CP[***" <TYPE-LIST-CP-FEATURE> "***]***" <NL>
{<TAB> <COMMENT>}*
[<CONSTRAINT-CARDINALITY>]
[<CONSTRAINT-GEOMETRY>]
{<SPACE4> <COMMENT>}*
<NL>
<TYPE-FIXING> ::=
<IDENT> " = " "***CP[***" <TYPE-LIST-CP-FEATURE> "***]***" <NL>
{<TAB> <COMMENT>}*
[<CONSTRAINT-CARDINALITY>]
[<CONSTRAINT-FIXING>]
{<SPACE4> <COMMENT>}*
<NL>

<TYPE-LIST-CP-ELEMENTARY> ::=
"***type:***" <TYPE-PART> "**,**" <NL> <TAB>
<TYPE-EXPR-CP-ELEM>
{"**,**" <NL> {<TAB> <COMMENT>}* <TAB> <TYPE-EXPR-CP-ELEM>}*
<TYPE-EXPR-CP-ELEM> ::=
<IDENT> "**:**" <TYPE-EXPR-ELEMENTARY>
<TYPE-EXPR-ELEMENTARY> ::=
<DATA-TYPE-NAME> ["*****"]
| "***SET[***" <DATA-TYPE-NAME> "***]***"
<DATA-TYPE-NAME> ::=
<IDENT>
| <PDEF-TYPE-NAME>

<TYPE-LIST-CP-ROOT> ::=
<TYPE-EXPR-CP-ELEM>
{"**,**" <NL> {<TAB> <COMMENT>}* <TAB> <TYPE-EXPR-CP-ELEM>}*
"**,**" <NL> {<TAB> <COMMENT>}* <TAB> "***type:ENUM[***" <CONST-VALUE-LIST> "***]***"
{"**,**" <NL> {<TAB> <COMMENT>}* <TAB> <TYPE-EXPR-CP-ELEM>}*
| "***type:ENUM[***" <CONST-VALUE-LIST> "***]***"
{"**,**" <NL> {<TAB> <COMMENT>}* <TAB> <TYPE-EXPR-CP-ELEM>}$^{+}$
<CONST-VALUE-LIST> ::=
<IDENT> {"**,**" <IDENT>}*

<TYPE-LIST-CP-LEAF> ::=
"***type:***" <TYPE-PART> "***,***" <NL> <TAB>
"***isA:***" <IDENT>

<TYPE-LIST-CP-FEATURE> ::=
"***type:***" <TYPE-FEATURE> "***,***" <NL> <TAB>
<TYPE-EXPR-CP-FEATURE>
{"***,***" <NL> {<TAB> <COMMENT>}* <TAB> <TYPE-EXPR-CP-FEATURE>}*
<TYPE-EXPR-CP-FEATURE> ::=
<IDENT> "***:***" <TYPE-EXPR-FEATURE>
<TYPE-EXPR-FEATURE> ::=
<DATA-TYPE-NAME> ["*******"]
| "***SET[***" <DATA-TYPE-NAME> "***]***"
| "***BAG[***" <DATA-TYPE-NAME> "***]***"
| "***SEQ[***" <DATA-TYPE-NAME> "***]***"

<CONSTRAINT-IDENTITY> ::=
<SPACE4> <CONSTRAINT-IDENTITY-TITLE>
<SPACE4> "***\WITH \ForAll p/***" <IDENT> " ***\and***" " ***\ForAll q/***"<IDENT> " ***:*** "
"***p <> q =>*** " <IDENT> "***(p)***" " ***<>*** " <IDENT> "***(q)***"
{" ***\or*** " "<IDENT> "***(p)***" " ***<>*** " "<IDENT> "***(q)***"}*
<NL>
<CONSTRAINT-COMPONENT> ::=
<SPACE4> <CONSTRAINT-COMPONENT-TITLE>
<SPACE4> "***\WITH \ForAll p/***" <IDENT> " ***:*** " <COMPONENT> <NL>
<CONSTRAINT-CONTAIN> ::=
<SPACE4> <CONSTRAINT-CONTAIN-TITLE>
<SPACE4> "***\WITH \ForAll p/***" <IDENT> " ***:*** " <COMPONENT> <NL>
<COMPONENT> ::=
("***Card(***" <IDENT> "***)*** " [">"] "= " ("***1***" | "***2***")
{" ***\or Card(***" <IDENT> "***)*** " [">"] "= " ("***1***" | "***2***")}*
| "***(Card(***" <IDENT> "***)***" {" ***+ Card(***" <IDENT> "***)***" }* "***) >= 2***")
<CONSTRAINT-CARDINALITY> ::=
<SPACE4> <CONSTRAINT-CARDINALITY-TITLE>
<SPACE4> "***\WITH \ForAll p/***" <IDENT> " ***:*** "
<CARDINALITY> <NL>
{<TAB> <SPACE4> "***\and*** " <CARDINALITY> <NL>}*
<CARDINALITY> ::=
"***Card(***" <IDENT> "***)*** " <OP> <SPACE> <NUMBER>
<CONSTRAINT-CONTIGUITY> ::=
<SPACE4> <CONSTRAINT-CONTIGUITY-TITLE>
<SPACE4> "***\WITH \ForAll p/***" <IDENT> " ***:*** " <FIXING> <NL>
<CONSTRAINT-ENUMERATION> ::=
<SPACE4> <CONSTRAINT-ENUMERATION-TITLE>
<SPACE4> "***\WITH \ForAll p :*** " <IDENT> <NL>
{<TAB> <SPACE4> "***(Type(p)=***" <IDENT> [" <=> " <ECC>] "***)***" " ***\and***" <NL>}*
<TAB> <SPACE4> "***(Type(p)=***" <IDENT> [" <=> " <ECC>] "***)***" <NL>

<ECC> ::=
<ENUMERATION>
{" **and** " <ENUMERATION>}*
[" **and** " <COMPONENT>]
{" **and** " <CARDINALITY>}*
[" **and** " <FIXING>]
<ENUMERATION> ::=
<IDENT> (" <> " | " = ") "**undef**"
<CONSTRAINT-GEOMETRY> ::=
<SPACE4> <CONSTRAINT-GEOMETRY-TITLE>
<SPACE4> "**WITH \ForAll p/**" <IDENT> " **:** " <COMPONENT> <NL>
<CONSTRAINT-FIXING> ::=
<SPACE4> <CONSTRAINT-FIXING-TITLE>
<SPACE4> "**WITH \ForAll p/**" <IDENT> " **:** " <FIXING> <NL>
<FIXING> ::=
"***Card(***" <IDENT> "***)*** " [">"] "**= *1***" {" **\or *Card(***" <IDENT> "***)*** " [">"] "**= *1***"}*

<OP> ::=
">=" | "<=" | "="

<STRING> ::=
{<CHAR>}$^{+}$
<CHAR> ::=
tout caractère imprimable excepté <NL>
<CHAR-NOT-SPACE> ::=
tout caractère imprimable exceptés <NL> et <SPACE>
<NL> ::=
nouvelle ligne de caractères
<SPACE4> ::=
<SPACE> <SPACE> <SPACE> <SPACE>
<SPACE> ::=
caractère espace
<TAB> ::=
caractère de tabulation
<COMMENT> ::=
"//" [<STRING>] <NL>
<IDENT> ::=
<CHAR-NOT-SPACE> [[<STRING>] <CHAR-NOT-SPACE>]
<PDEF-TYPE-NAME> ::=
"***STRING***" | "***CHAR***" | "***BOOLEAN***" | "***INTEGER***" | "***RATIONAL***" | "***DURATION***"
<TYPE-PART> ::=
"***BasicPart***" | "***CompoundPart***" | "***PileOfParts***" | "***Buffer***" | "***Container***"
<TYPE-FEATURE> ::=
"***PartPhysicalFeature***" | "***PartGeometricalFeature***" | "***PartFixingFeature***" |
"***PartContiguityFeature***" | "***PartEnclosureFeature***"
<NUMBER> ::=
(1-9) {(0-9)}*

<CONST-ELEMENTARY-TITLE> ::=
"***//CLASSES DE PARTS ÉLÉMENTAIRES***" <NL> <NL>
<CONST-ROOT-TITLE> ::=
"***//SUPER CLASSES DE PARTS RACINES***" <NL> <NL>
<CONST-SUB-CLASS-TITLE> ::=
"***//CLASSES DE PARTS SOUS-ENSEMBLES ET SUPER CLASSES***" <NL> <NL>
<CONST-LEAF-TITLE> ::=
"***//SOUS-CLASSES DE PARTS FEUILLES***" <NL> <NL>
<CONST-PHYSICAL-TITLE> ::=
"***//CLASSES DE DISPOSITIFS PHYSIQUES***" <NL> <NL>
<CONST-GEOMETRICAL-TITLE> ::=
"***//CLASSES DE DISPOSITIFS GÉOMÉTRIQUES***" <NL> <NL>
<CONST-FIXING-TITLE> ::=
"***//CLASSES DE DISPOSITIFS DE FIXATION***" <NL> <NL>
<CONST-CONTIGUITY-TITLE> ::=
"***//CLASSES DE DISPOSITIFS DE CONTIGUÏTÉ***" <NL> <NL>
<CONST-ENCLOSURE-TITLE> ::=
"***//CLASSES DE DISPOSITIFS D'ENCLOS***" <NL> <NL>
<CONSTRAINT-IDENTITY-TITLE> ::=
"***//Contrainte d'identité***" <NL>
<CONSTRAINT-COMPONENT-TITLE> ::=
"***//Contrainte de composition***" <NL>
<CONSTRAINT-CONTAIN-TITLE> ::=
"***//Contrainte de contenance***" <NL>
<CONSTRAINT-CARDINALITY-TITLE> ::=
"***//Contrainte de cardinalité***" <NL>
<CONSTRAINT-CONTIGUITY-TITLE> ::=
"***//Contrainte de contiguïté***" <NL>
<CONSTRAINT-ENUMERATION-TITLE> ::=
"***//Contrainte d'énumération***" <NL>
<CONSTRAINT-GEOMETRY-TITLE> ::=
"***//Contrainte de géométrie***" <NL>
<CONSTRAINT-FIXING-TITLE> ::=
"***//Contrainte de fixation***" <NL>

<HDR-TITLE> ::=
"***%SPEC***"
<HDR-BASIC-TYPES> ::=
"***%BASIC TYPES***" <NL> <NL>
<HDR-CONSTR-TYPES> ::=
"***%CONSTRUCTED TYPES***" <NL> <NL>

Annexe D

Exemple de modélisation graphique de "parts"

Pour illustrer les concepts liés au sous-ensemble du pattern *Part*, nous avons choisi de présenter un exemple complet et cohérent de modélisation graphique, ainsi que sa traduction en langage AlbertII. Cet exemple est inspiré de la nouvelle série 6400 des tracteurs Massey Ferguson. Nous avons retenu trois modèles :

- le MF 6470 de 125 chevaux avec un moteur 4 cylindres de 4,4 litres;
- le MF 6490 de 170 chevaux avec un moteur 6 cylindres de 6,6 litres;
- le MF 6499 de 215 chevaux avec un moteur 6 cylindres de 7,4 litres.

Figure D.1 : Exemple de tracteur Massey Ferguson

Cette annexe comporte quatre parties. La première présente une description en langage courant des composants et des caractéristiques des trois modèles cités ci-dessus. La deuxième contient un tableau reprenant pour chaque composant ou caractéristique la classe de parts ou d'états qui le représente au sein du modèle et les classes d'attributs auxquelles cette classe est éventuellement associée. Tous les types de classes ont été employés à l'exception de la classe de tampons et de la classe d'attributs contenus qui sont plutôt réservées aux machines de production. La troisième donne une représentation graphique des trois modèles de tracteur au moyen de différents schémas classés par thèmes. Cette représentation est conforme aux contraintes vérifiées par l'éditeur Visio. La dernière partie propose la traduction de la modélisation graphique en langage AlbertII. Il s'agit de la traduction mot pour mot générée par l'éditeur Visio.

D.1. Présentation des trois modèles de tracteur

Les trois modèles de la série 6400 de chez Massey Ferguson disposent d'une plate-forme commune et se différencient essentiellement par la puissance, la taille ou le caractère optionnel de certains composants.

Dans un tracteur, on distingue généralement, d'une part, le châssis et, d'autre part, la carrosserie et une série d'organes externes. Le châssis est formé d'un cadre sur lequel viennent se greffer sept éléments qui sont le train de roulement, le moteur, l'équipement électrique, la transmission (dont la boîte de vitesse), la prise de force (arrière) et les systèmes hydraulique et de relevage. Le train de roulement est composé du pont avant et du pont arrière (ou pont moteur) qui reçoit le mouvement de la transmission et qui le communique aux roues motrices (tous les modèles possèdent les quatre roues motrices). Chaque pont dispose d'un essieu et de deux roues (celles du pont avant étant plus petites). Le pont arrière comprend en outre une suspension faite de lames de ressort en acier empilées les unes sur les autres.

Les trois modèles de tracteur possèdent un moteur diesel 4 temps à injection mais de puissance différente. Ce dernier est constitué d'un équipage fixe, d'un équipage mobile et d'accessoires. L'équipage fixe se compose du bloc-moteur (4 cylindres de 1,1 litre pour le MF 6470, 6 cylindres de 1,1 litre pour le MF 6490 et 6 cylindres de 1,23 litre pour le MF 6499) où se logent les pistons et où se fixent l'équipage mobile et les accessoires, de la culasse qui referme le tout et du carter à huile qui referme le bas. L'équipage mobile est constitué du vilebrequin, des bielles, des pistons (le MF 6499 disposant de plus gros pistons que les deux autres modèles), des segments et de la distribution. Chaque bielle est soudée à son piston. Quant aux accessoires, ils comprennent la pompe à huile, la pompe à eau, l'alternateur et le démarreur.

Le débit maximal du système hydraulique est de 129 litres/minute pour le MF 6470, de 158 litres/minute pour le MF 6490 et de 165 litres/minute pour le MF 6499. La capacité de relevage est de 6730 kilos pour le MF 6470, de 9100 kilos pour le MF 6490 et de 9570 kilos pour le MF 6499.

La cabine, le capot, les ailes, la grille avant et le garde-boue arrière constituent la carrosserie. Tous les éléments de la carrosserie sont peints en rouge sauf la grille avant et la cabine qui sont de couleurs noire et argentée. La grille du MF 6499 comprend également un ventilateur à 6 palmes. Le nom de la firme et le numéro de série du tracteur sont inscrits sur chacune des deux ailes. La suspension pneumatique de la cabine est réglable aux quatre coins. Celle-ci est boulonnée sur trois côtés du garde-boue. Pour faciliter la réparation et l'entretien du moteur, le capot est simplement vissé aux autres éléments de la carrosserie : l'arrière du capot est fixé à l'avant de la cabine, l'avant du capot est rivé au sommet de la grille, tandis que les côtés du capot sont vissés sur la partie supérieure des deux ailes.

Le tracteur contient également un réservoir à mazout placé sous la cabine et d'une contenance de 200 litres pour le plus petite modèle et de 380 litres pour les deux autres. Pour prévenir les grincements et les claquements, les cognements et les cliquetis, ainsi que les vibrations, des coussins (ou isolateurs) en divers matériaux, comme le caoutchouc synthétique, sont installés entre le réservoir, la carrosserie et le châssis.

La climatisation, le contrôle de patinage, le système d'information et de contrôle Datatronic III, ainsi que l'attelage et la prise de force avant sont de série sur le gros modèle, en option sur le modèle intermédiaire et non disponible sur le plus petite modèle.

La firme Massey Ferguson offre en promotion un jeu de douze poids (6 légers au milieu et 3 lourds de chaque côté) à glisser des deux côtés de l'attache avant du tracteur.

D.2. Tableau des classes

La première colonne du tableau contient un composant ou la caractéristique d'une pièce ou d'un ensemble de pièces du tracteur, la deuxième le type de classes qui représente ce composant ou cette caractéristique, la troisième le nom de cette classe et la quatrième les types de classes d'attributs qu'elle possède accompagnés de leurs cardinalités et des éléments de destination.

Le tableau ne contient pas le nom des classes d'attributs et ne répertorie par les classes d'identités ou de dispositifs de valeur dont le type est prédéfini.

La cardinalité n'est pas toujours indiquée si celle-ci équivaut à "1-1".

Composant / caractéristique	**Type de classes**	**Nom de la classe**	**Classes d'attributs**
Tracteur	Part composée	*Tracteur*	Identités : nombre entier Identités : chaîne de caractères Positions : bâtiment Positions : atelier du bâtiment Composants : 1 châssis Composants : 1 carrosserie Composants : 0-1 réservoir Composants : 0-12 poids Dispositif physique : 1 climatisation Dispositif physique : 1 contrôle de patinage Dispositif physique : 1 Datatronic III Dispositif physique : 1 attelage et prise de force avant Géométries : 1 dispositif
Bâtiment	Position	*Bâtiment*	—
Atelier	Position	*Atelier*	—
Châssis	Part composée	*Châssis*	Identités : numéro de châssis Composants : 1 cadre Composants : 0-1 train de roulement Composants : 0-1 moteur Composants : 0-1 transmission Composants : 0-1 équipement électrique Composants : 0-1 prise de force Composants : 0-1 système hydraulique Composants : 0-1 système de relevage
Numéro de châssis	Identité	*Numéro_châssis*	—
Cadre	Part de base	*Cadre*	Identités : nombre entier
Train de roulement	Part composée	*Train_de_ roulement*	Identités : nombre entier Composants : 1 pont avant Composants : 1 pont arrière
Pont avant	Part composée	*Pont_avant*	Identités : nombre entier Composants : 0-1 essieu Composants : 0-2 petites roues

Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Pont arrière	Part composée	*Pont_arrière*	Identités : nombre entier Composants : 0-1 essieu Composants : 0-2 grosses roues Composants : 0-1 suspension
Essieu	Part de base	*Essieu*	Identités : nombre entier
Petite roue	Part de base	*Roue_avant*	Identités : nombre entier
Grosse roue	Part de base	*Roue_arrière*	Identités : nombre entier
Suspension	Pile LIFO	*Suspension*	Identités : nombre entier Composants : 2-10 lames de ressort Contiguïtés : 1-9 dispositifs Enclos : 1 dispositif
Lames de ressort	Part de base	*Lame*	Identités : nombre entier Dispositifs physiques : métal (acier)
Lame en acier	Dispositif physique	*Acier*	—
Lames de ressort contiguës	Dispositif de contiguïté	*Contiguïté_lames*	Dispositifs de contiguïté : 2 lames
Commentaire sur la contiguïté des lames	Commentaire	—	"Lames contiguës deux à deux"
Lames de ressort encloses	Dispositif d'enclos	*Enclos_lames*	Dispositifs d'enclos : 1-9 lames Valeurs : nombre de lames en dessous Valeurs : nombre de lames au-dessus
Commentaire sur l'enclosure des lames	Commentaire	—	"Lame du bas enclose par l'essieu"
Moteur	Part composée	*Moteur*	Identités : numéro de moteur Composants : 1 équipage fixe Composants : 1 équipage mobile Composants : 0-1 accessoires
Commentaire sur le type de moteur	Commentaire	—	"Moteur diesel 4 temps à injection"
Commentaire sur la puissance du MF 6470	Commentaire	—	"MF 6470 de 125 chevaux avec un moteur 4 cylindres de 4,4 litres"
Commentaire sur la puissance du MF 6490	Commentaire	—	"MF 6490 de 170 chevaux avec un moteur 6 cylindres de 6,6 litres"
Commentaire sur la puissance du MF 6499	Commentaire	—	"MF 6499 de 215 chevaux avec un moteur 6 cylindres de 7,4 litres"
Numéro de moteur	Identité	*Numéro_moteur*	—
Équipage fixe	Part composée	*Équipage_fixe*	Identités : nombre entier Composants : 1 bloc moteur Composants : 0-1 culasse Composants : 0-1 carter à huile
Bloc moteur	Part de base (racine)	*Bloc_moteur*	Identités : nombre entier Dispositifs physiques : modèle (MF 64..)
Bloc moteur de 4 cylindres de 1,1 litre	Part de base (feuille)	*Bloc_moteur_4_1,1*	Sous-classes : bloc moteur

Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Commentaire sur le moteur du MF 6470	Commentaire	—	"125 chevaux"
Bloc moteur de 6 cylindres de 1,1 litre	Part de base (feuille)	*Bloc_moteur_6_1,1*	Sous-classes : bloc moteur
Commentaire sur le moteur du MF 6490	Commentaire	—	"170 chevaux"
Bloc moteur de 6 cylindres de 1,23 litre	Part de base (feuille)	*Bloc_moteur_6_1,23*	Sous-classes : bloc moteur
Commentaire sur le moteur du MF 6499	Commentaire	—	"215 chevaux"
Bloc moteur de tel modèle de tracteur	Dispositif physique	*Modèle_tracteur*	—
Culasse	Part de base	*Culasse*	Identités : nombre entier
Carter à huile	Part de base	*Carter*	Identités : nombre entier
Équipage mobile	Part composée (racine)	*Équipage_mobile*	Identités : nombre entier Composants : 0-1 vilebrequin Composants : 0-1 segments Composants : 0-1 distribution
Équipage mobile du MF 6470	Part composée (feuille)	*Équipage_mobile_MF6470*	Sous-classes : équipage mobile Composants : 0-4 bielles Composants : 0-4 petits pistons Géométries : 0-4 dispositifs
Équipage mobile des modèles supérieurs	Part composée (super et sous-classe)	*Équipage_mobile_modèles_supérieurs*	Sous-classes : équipage mobile Composants : 0-6 bielles
Équipage mobile du MF 6490	Part composée (feuille)	*Équipage_mobile_MF6490*	Sous-classes : équipage mobile des modèles supérieurs Composants : 0-6 petits pistons Géométries : 0-6 dispositifs
Équipage mobile du MF 6499	Part composée (feuille)	*Équipage_mobile_MF6499*	Sous-classes : équipage mobile des modèles supérieurs Composants : 0-6 gros pistons Géométries : 0-6 dispositifs
Bielle	Part de base	*Bielle*	Identités : nombre entier
Petit piston	Part de base	*Piston_petit*	Identités : nombre entier
Gros piston	Part de base	*Piston_gros*	Identités : nombre entier
Bielle et piston du MF 6470 assemblés	Dispositif géométrique	*Assemblage_bielle_piston_MF6470*	Dispositifs géométriques : 0-1 bielle Dispositifs géométriques : 0-1 petit piston
Bielle et piston du MF 6490 assemblés	Dispositif géométrique	*Assemblage_bielle_piston_MF6490*	Dispositifs géométriques : 0-1 bielle Dispositifs géométriques : 0-1 petit piston
Bielle et piston du MF 6499 assemblés	Dispositif géométrique	*Assemblage_bielle_piston_MF6499*	Dispositifs géométriques : 0-1 bielle Dispositifs géométriques : 0-1 gros piston
Bielle et piston du MF 6470 soudés	Dispositif de fixation	*Fixation_bielle_piston_MF6470*	Dispositifs de fixation : 0-4 dispositifs Valeurs : soudure

Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Bielle et piston du MF 6490 soudés	Dispositif de fixation	*Fixation_bielle_ piston_MF6490*	Dispositifs de fixation : 0-6 dispositifs Valeurs : soudure
Bielle et piston du MF 6499 soudés	Dispositif de fixation	*Fixation_bielle_ piston_MF6499*	Dispositifs de fixation : 0-6 dispositifs Valeurs : soudure
Bielle et piston soudés	Dispositif de valeur	*Soudure*	—
Vilebrequin	Part de base	*Vilebrequin*	Identités : nombre entier
Segments	Part de base	*Segments*	Identités : nombre entier
Distribution	Part de base	*Distribution*	Identités : nombre entier
Pistons enclos par le bloc moteur du MF 6470	Conteneur	*Conteneur_bloc_ moteur_MF6470*	Identités : nombre entier Contenants : 0-4 petits pistons Contenants : 1 bloc moteur MF 6470 Contiguïtés : 1-4 dispositifs Enclos : 1 dispositif
Pistons enclos par le bloc moteur du MF 6490	Conteneur	*Conteneur_bloc_ moteur_MF6490*	Identités : nombre entier Contenants : 0-6 petits pistons Contenants : 1 bloc moteur MF 6490 Contiguïtés : 1-6 dispositifs Enclos : 1 dispositif
Pistons enclos par le bloc moteur du MF 6499	Conteneur	*Conteneur_bloc_ moteur_MF6499*	Identités : nombre entier Contenants : 0-6 gros pistons Contenants : 1 bloc moteur MF 6499 Contiguïtés : 1-6 dispositifs Enclos : 1 dispositif
Petit piston contigu au bloc moteur MF 6470	Dispositif de contiguïté	*Contiguïté_bloc_ moteur_MF6470*	Dispositifs de contiguïtés : 1 petit piston Dispositifs de contiguïtés : 1 bloc moteur MF 6470
Petit piston contigu au bloc moteur MF 6490	Dispositif de contiguïté	*Contiguïté_bloc_ moteur_MF6490*	Dispositifs de contiguïtés : 1 petit piston Dispositifs de contiguïtés : 1 bloc moteur MF 6490
Gros piston contigu au bloc moteur MF 6499	Dispositif de contiguïté	*Contiguïté_bloc_ moteur_MF6499*	Dispositifs de contiguïtés : 1 gros piston Dispositifs de contiguïtés : 1 bloc moteur MF 6499
Petits pistons enclos par bloc moteur MF 6470	Dispositif d'enclos	*Enclos_bloc_ moteur_MF6470*	Dispositifs d'enclos : 0-4 petits pistons
Petits pistons enclos par bloc moteur MF 6490	Dispositif d'enclos	*Enclos_bloc_ moteur_MF6490*	Dispositifs d'enclos : 0-6 petits pistons
Gros pistons enclos par bloc moteur MF 6499	Dispositif d'enclos	*Enclos_bloc_ moteur_MF6499*	Dispositifs d'enclos : 0-6 gros pistons
Accessoires	Part composée	*Accessoires*	Identités : nombre entier Composants : 0-1 pompe à huile Composants : 0-1 pompe à eau Composants : 0-1 alternateur Composants : 0-1 démarreur
Pompe à huile	Part de base	*Pompe_huile*	Identités : nombre entier

Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Pompe à eau	Part de base	*Pompe_eau*	Identités : nombre entier
Alternateur	Part de base	*Alternateur*	Identités : nombre entier
Démarreur	Part de base	*Démarreur*	Identités : nombre entier
Culasse et carter contigus	Conteneur	*Conteneur_ culasse_carter*	Identités : nombre entier Contenants : 0-1 culasse Contenants : 0-1 carter Contiguïtés : 1 dispositif
Culasse contiguë au carter	Dispositif de contiguïté	*Contiguïté_ culasse_carter*	Dispositifs de contiguïté : 1 culasse Dispositifs de contiguïté : 1 carter
Transmission	Part de base	*Transmission*	Identités : nombre entier
Équipement électrique	Part de base	*Équipement_ électrique*	Identités : nombre entier
Prise de force	Part de base	*Prise_de_force*	Identités : nombre entier
Système hydraulique	Part de base	*Système_ hydraulique*	Identités : nombre entier Dispositifs physiques : débit
Système de relevage	Part de base	*Système_de_ relevage*	Identités : nombre entier Dispositifs physiques : capacité
Débit	Dispositif physique	*Débit*	Valeurs : nombre entier "Débit : 129 - 158 - 165 litres/minute"
Capacité de relevage	Dispositif physique	*Capacité_ relevage*	Valeurs : nombre entier "6730 kilos - 9100 kilos - 9570 kilos"
Carrosserie	Part composée	*Carrosserie*	Identités : nombre entier Composants : 0-1 cabine Composants : 0-1 capot Composants : 0-2 ailes Composants : 0-1 grille avant Composants : 0-1 garde-boue Géométries : 0-3 dispositifs (cabine) Géométries : 0-1 dispositif (capot) Géométries : 0-1 dispositif (grille) Géométries : 0-2 dispositifs (ailes)
Cabine	Part de base	*Cabine*	Identités : nombre entier Dispositifs physiques : 0-2 couleur Dispositifs physiques : suspension pneumatique
Capot	Part de base	*Capot*	Identités : nombre entier Dispositifs physiques : 0-1 couleur
Aile	Part de base	*Aile*	Identités : nombre entier Dispositifs physiques : 0-1 couleur Dispositifs physiques : type d'aile Dispositifs physiques : nom firme Dispositifs physiques : numéro série
Grille avant	Part mixte	*Grille_avant*	Identités : nombre entier Dispositifs physiques : 0-2 couleurs
Grille des modèles inférieurs	Part de base	*Grille_simple*	Sous-classes : grille avant

Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Grille du MF 6499	Part composée	*Grille_composée*	Sous-classes : grille avant Composants : 1 grosse grille Composants : 1 ventilateur
Grosse grille du MF 6499	Part de base	*Grille_grosse*	Identités : nombre entier
Ventilateur	Part de base	*Ventilateur*	Identités : nombre entier Dispositifs physiques : avec palmes
Type de ventilateur	Dispositif physique	*Type_ventilateur*	Valeurs : 6 palmes identiques et ordonnées
Ventilateur à 6 palmes identiques et ordonnées	Dispositif de valeur	*Palme*	—
Garde-boue arrière	Part de base	*Garde_boue*	Identités : nombre entier Dispositifs physiques : 0-1 couleur
Couleur	Dispositif physique	*Couleur*	Valeurs : rouge, noir ou argenté
Type de couleur	Dispositif de valeur	*Rouge/noir/ argenté*	—
Type d'aile	Dispositif physique	*Type_aile*	Valeurs : gauche ou droite
Aile gauche ou droite	Dispositif de valeur	*Gauche/Droite*	—
Nom de la firme	Dispositif physique	*Nom_firme*	—
Commentaire sur le nom de la firme	Commentaire	—	"Massey Ferguson"
Numéro de la série	Dispositif physique	*Numéro_série*	—
Commentaire sur le numéro de la série	Commentaire	—	"MF 6470 – MF 6490 – MF 6499"
Suspension pneumatique de cabine	Dispositif physique	*Suspension_ pneumatique*	Valeurs : 4 lettres (une lettre pour chacun des 4 côtés de la cabine)
Suspension : suite de 5 caractères selon la position	Dispositif de valeur	*Positions*	—
Commentaire sur la suspension	Commentaire	—	"Positions : a-b-c-d-e"
Cabine et garde-boue assemblés	Dispositif géométrique	*Assemblage_ cabine_garde_ boue*	Dispositifs géométriques : 0-1 cabine Dispositifs géométriques : 0-1 garde-boue Valeurs : type de côté (3 possibilités)
Type de côté du garde-boue	Dispositif de valeur	*Type_côté_ garde_boue*	—
Commentaire sur type de côté du garde-boue	Commentaire	—	"Côtés : arrière – gauche – droite"
Cabine et garde-boue boulonnés	Dispositif de fixation	*Fixation_cabine_ garde_boue*	Dispositifs de fixation : 0-3 dispositifs Valeurs : boulonnage

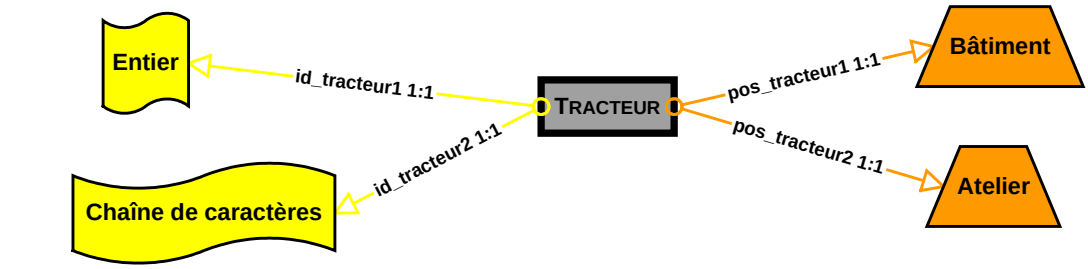
Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Commentaire sur la fixation de la cabine	Commentaire	—	"Forte fixation de la cabine au garde-boue pour éviter les décrochages."
Boulonnage de la cabine	Dispositif de valeur	*Boulonnage*	—
Capot et cabine assemblés	Dispositif géométrique	*Assemblage_ capot_cabine*	Dispositifs géométriques : 0-1 capot Dispositifs géométriques : 0-1 cabine Valeur : côté arrière du capot
Capot et grille assemblés	Dispositif géométrique	*Assemblage_ capot_grille*	Dispositifs géométriques : 0-1 capot Dispositifs géométriques : 0-1 grille Valeur : côté avant du capot
Capot et aile assemblés	Dispositif géométrique	*Assemblage_ capot_aile*	Dispositifs géométriques : 0-1 capot Dispositifs géométriques : 0-1 aile Valeur : côté gauche ou droit du capot Valeurs : aile gauche ou droite
Type de côté du capot	Dispositif de valeur	*Type_côté_capot*	—
Commentaire sur type de côté du capot	Commentaire	—	"Côtés : arrière – avant – gauche – droite"
Capot, cabine, grille et ailes vissés	Dispositif de fixation	*Fixation_capot*	Dispositifs de fixation : 0-1 dispositif (cabine) Dispositifs de fixation : 0-1 dispositif (grille) Dispositifs de fixation : 0-2 dispositifs (ailes) Valeurs : vissage
Vissage du capot	Dispositif de valeur	*Vissage*	—
Réservoir à mazout	Part de base	*Réservoir*	Identités : nombre entier Dispositifs physiques : capacité
Capacité du réservoir	Dispositif physique	*Capacité*	Valeurs : nombre entier "200 litres - 380 litres"
Réservoir, châssis et carrosserie isolés	Dispositif géométrique	*Assemblage_ isolation*	Dispositifs géométriques : 1 châssis Dispositifs géométriques : 1 carrosserie Dispositifs géométriques : 0-1 réservoir
Commentaire sur l'isolation du tracteur	Commentaire	—	"Pour prévenir les grincements et les claquements, les cognements et les cliquetis, ainsi que les vibrations, des coussins (ou isolateurs) en divers matériaux, comme le caoutchouc synthétique, sont installés entre le réservoir, la carrosserie et le châssis."

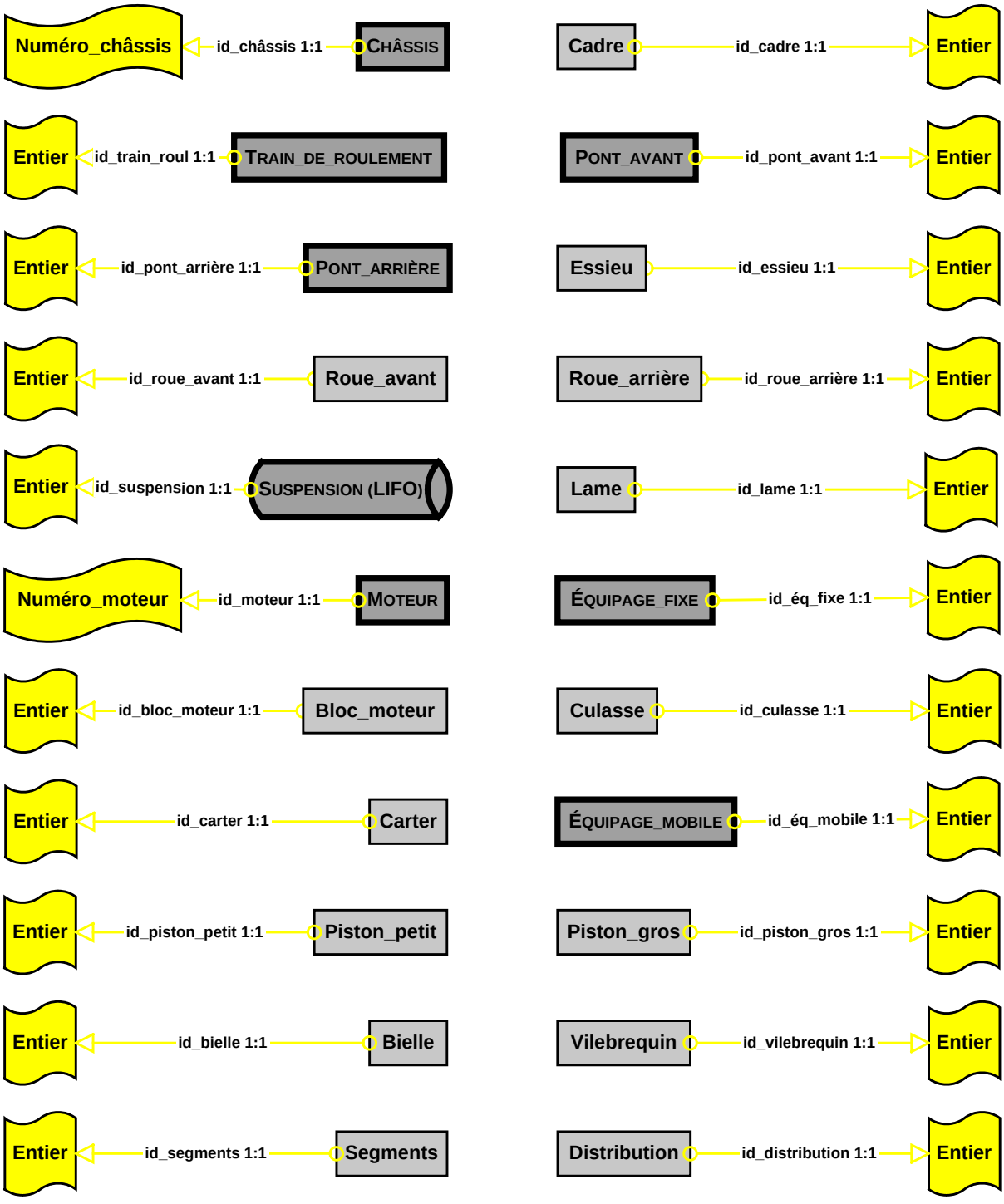
Composant / caractéristique	Type de classes	Nom de la classe	Classes d'attributs
Poids à l'avant	Pile standard	*Poids*	Identités : nombre entier Composants : 0-6 poids légers Composants : 0-6 poids lourds Contiguïtés : 0-5 dispositifs (légers) Contiguïtés : 0-4 dispositifs (lourds) Contiguïtés : 0-2 dispositifs (mixtes) Enclos : 1 dispositif
Commentaire sur les poids	Commentaire	—	"Promotion : offre gratuite d'un jeu de douze poids"
Poids léger	Part de base	*Poids_léger*	Identités : nombre entier
Poids lourd	Part de base	*Poids_lourd*	Identités : nombre entier
Poids légers contigus	Dispositif de contiguïté	*Contiguïté_ poids_légers*	Dispositifs de contiguïté : 2 poids légers
Poids lourds contigus	Dispositif de contiguïté	*Contiguïté_ poids_lourds*	Dispositifs de contiguïté : 2 poids lourds
Poids légers et lourds contigus	Dispositif de contiguïté	*Contiguïté_ poids_léger_lourd*	Dispositifs de contiguïté : 1 poids léger Dispositifs de contiguïté : 1 poids lourd
Poids légers et lourds enclos	Dispositif d'enclos	*Enclosure_poids*	Dispositifs d'enclos : 0-6 poids légers Dispositifs d'enclos : 0-4 poids lourds
Climatisation	Dispositif physique	*Climatisation*	Valeurs : de série, en option ou non disponible
Contrôle de patinage	Dispositif physique	*Contrôle_de_ patinage*	Valeurs : de série, en option ou non disponible
Système Datatronic III	Dispositif physique	*Datatronic_III*	Valeurs : de série, en option ou non disponible
Attelage et prise de force avant	Dispositif physique	*Attelage_prise_ de_force_avant*	Valeurs : de série, en option ou non disponible
Composant de série, en option ou non disponible	Dispositif de valeur	*Série/option/ non_disponible*	—

Tableau D.1 : Classes du modèle graphique représentant
les trois modèles de tracteurs Massey Ferguson

D.3. Représentation graphique

1. Les relations d'identités et de positions





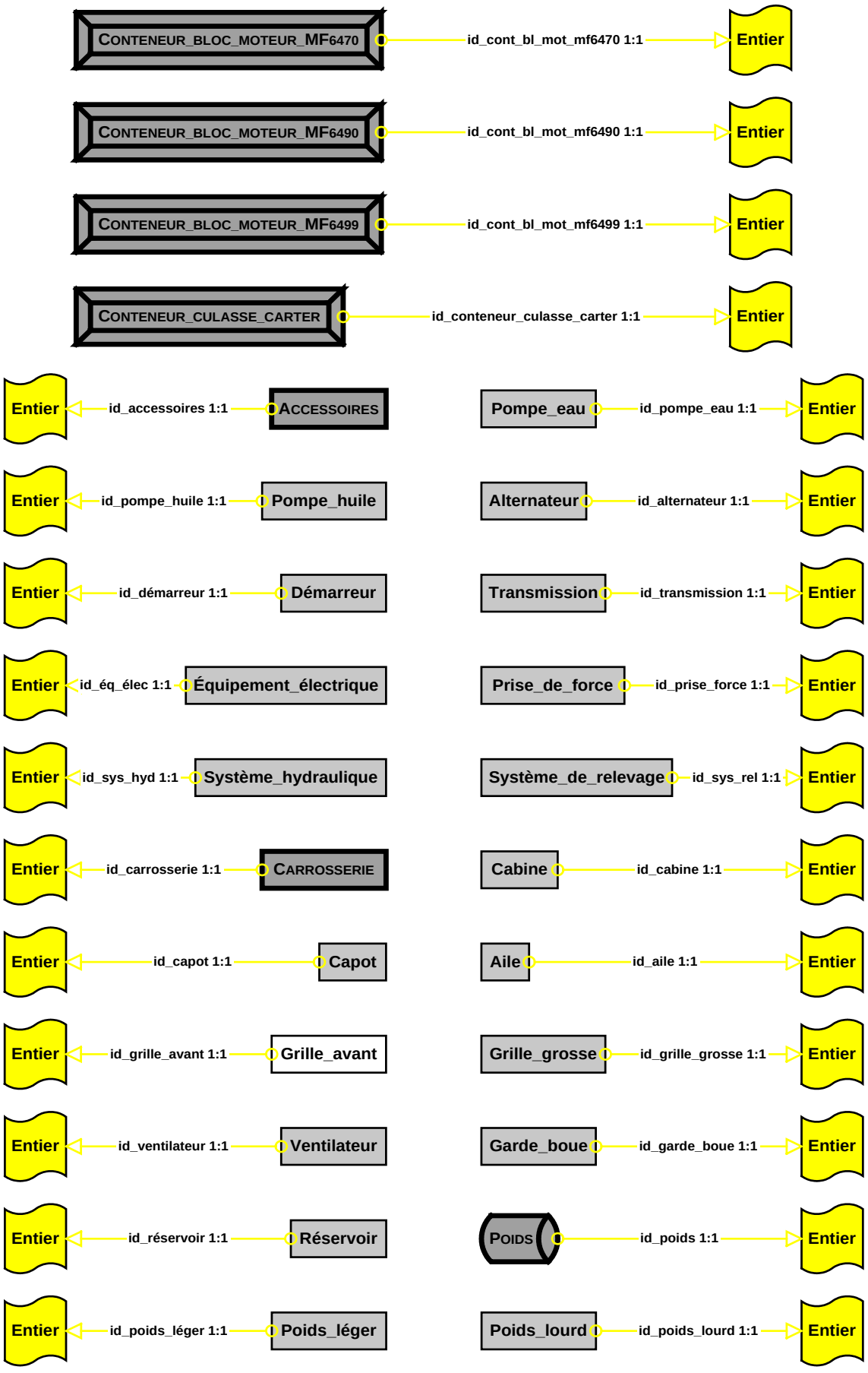


Figure D.2 : Classes d'identités et de positions du modèle de tracteurs Massey Ferguson

2. **Les relations de composition et de spécialisation**

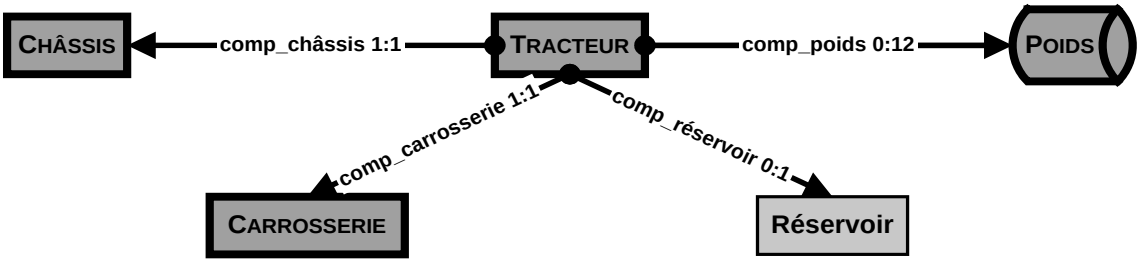


Figure D.3 : Classes de parts composantes de la classe de parts composées *Tracteur*

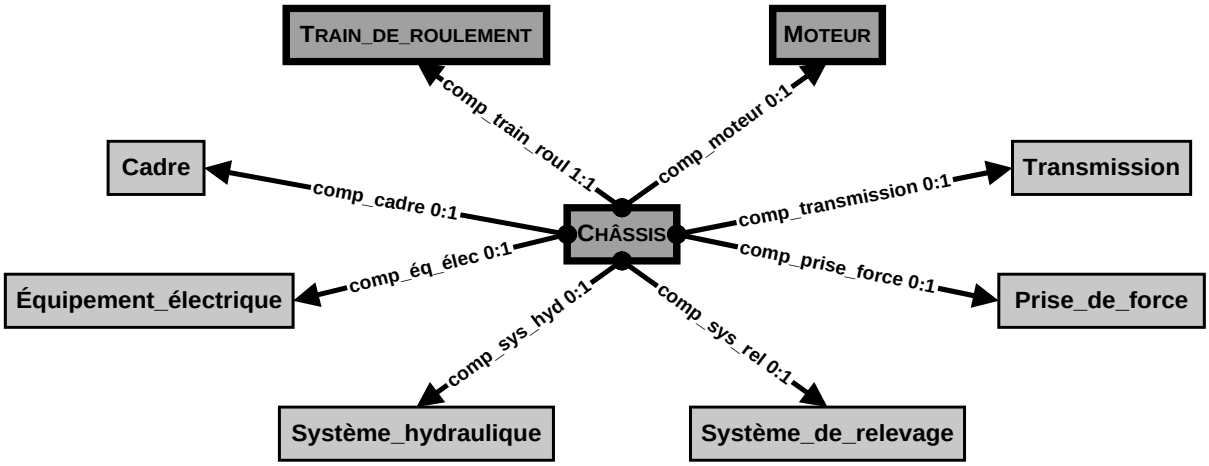


Figure D.4 : Classes de parts composantes de la classe de parts composées *Châssis*

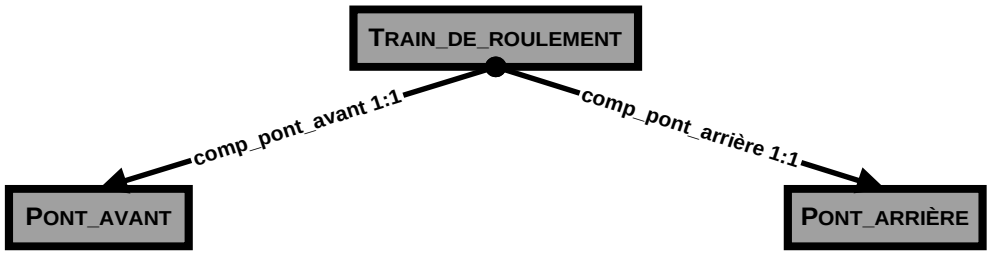


Figure D.5 : Classes de parts composantes de la classe de parts composées *Train_de_roulement*

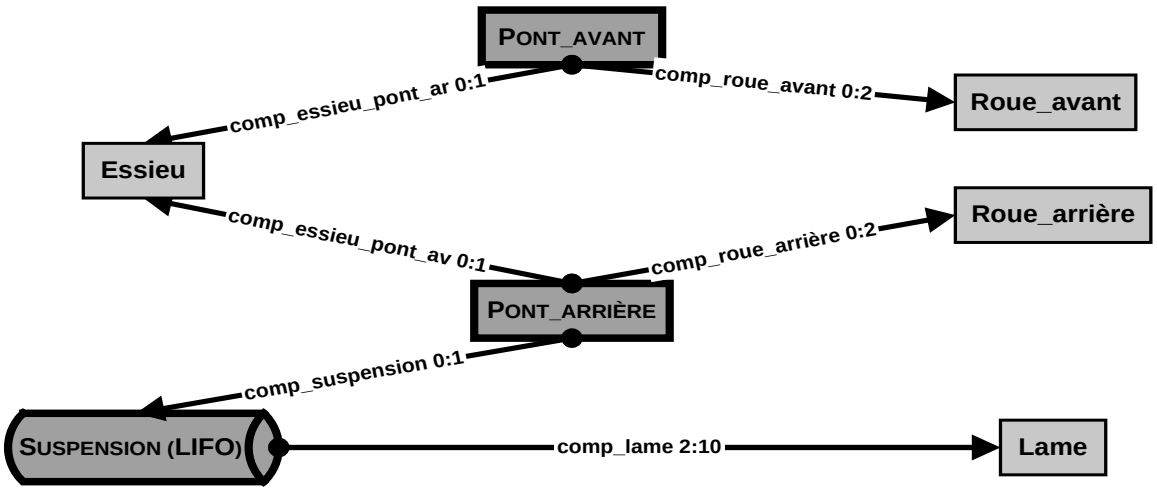


Figure D.6 : Classes de parts composantes des classes de parts composées *Pont_avant* et *Pont_arrière*

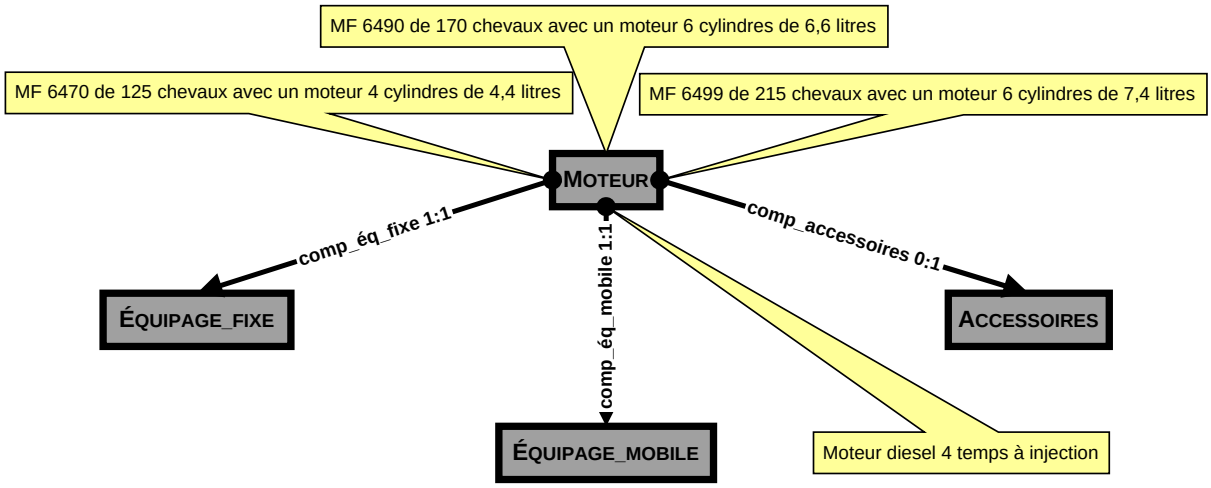


Figure D.7 : Classes de parts composantes de la classe de parts composées *Moteur*

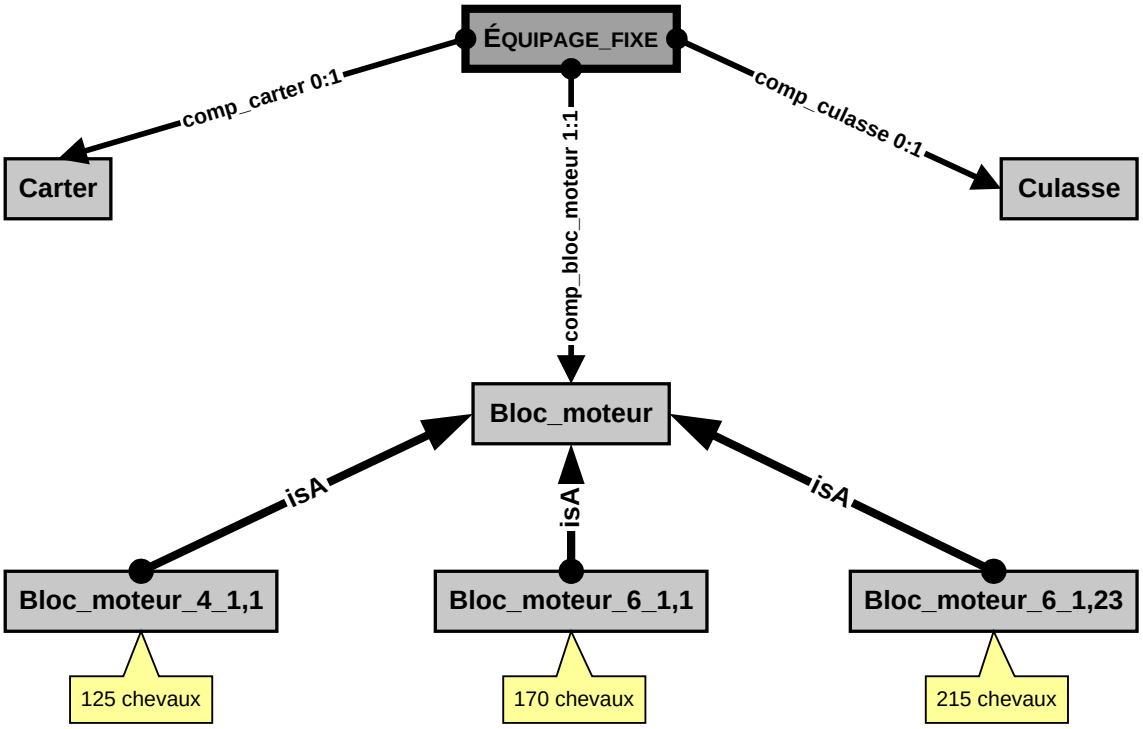


Figure D.8 : Classes de parts composantes de la classe de parts composées *Équipage_fixe* et hiérarchie de spécialisation de la classe de parts de base racine *Bloc_moteur*

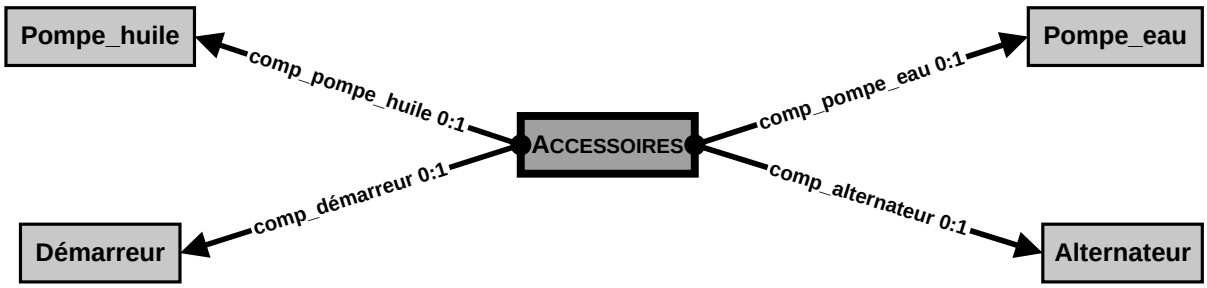


Figure D.9 : Classes de parts composantes de la classe de parts composées *Accessoires*

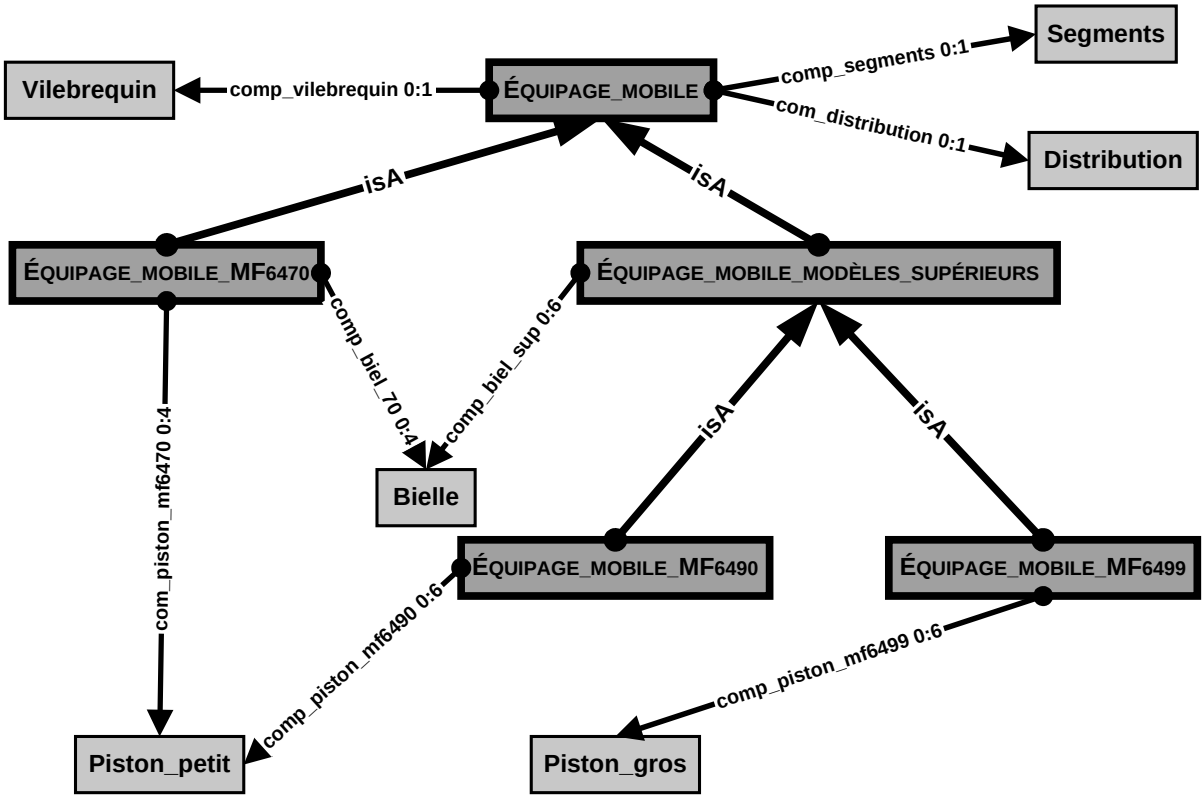


Figure D.10 : Hiérarchies de spécialisation et de composition de la classe *Équipage_mobile*

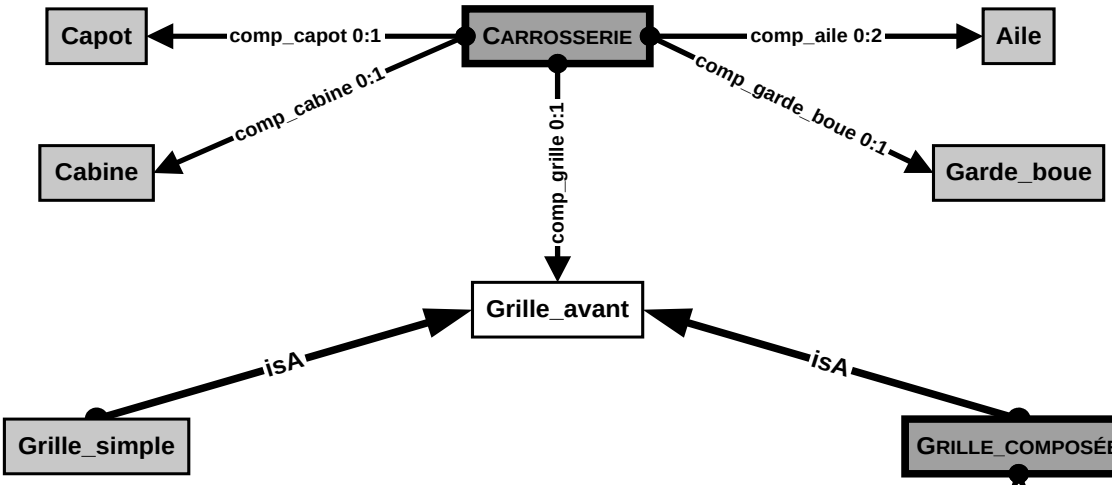


Figure D.11 : Classes de parts composantes de la classe de parts composées *Carrosserie*

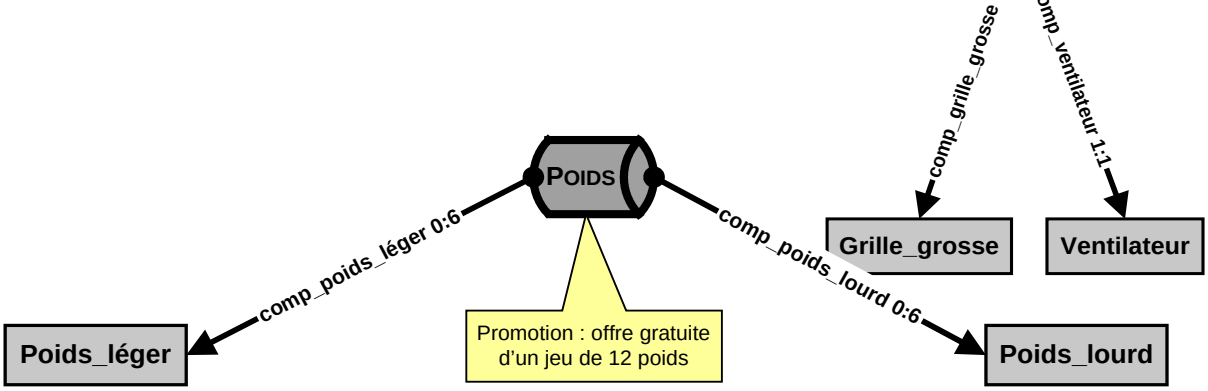


Figure D.12 : Classes de parts composantes de la classe de parts composées *Poids*

3. **Synthèse des relations de composition et de spécialisation**[53]

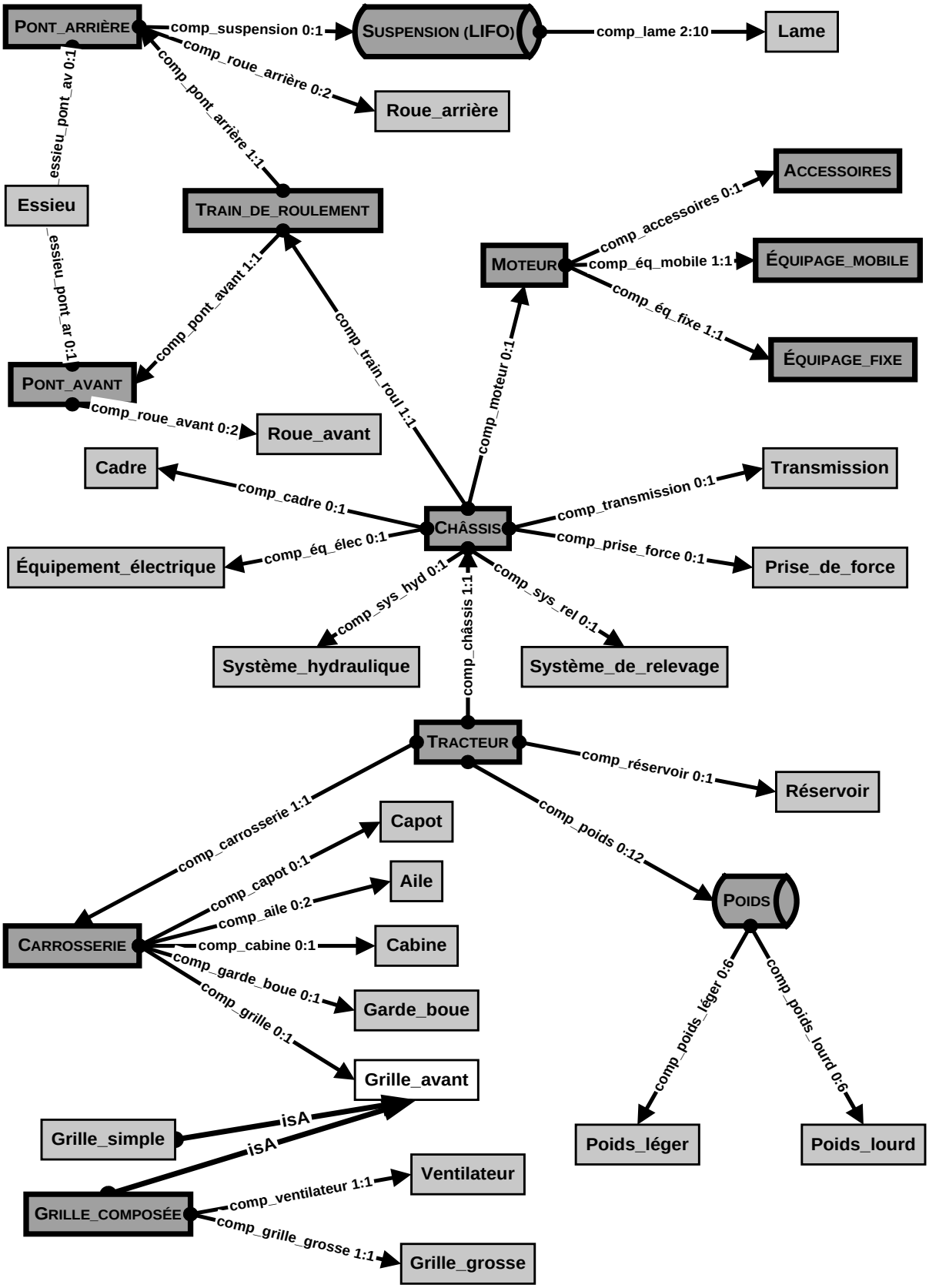


Figure D.13 : Hiérarchies de spécialisation et de composition de la classe *Tracteur*

[53] Par manque de place, les composants du moteur n'ont pu être développés ici. Pour un aperçu complet, voir les trois schémas relatifs à l'équipage fixe, aux accessoires et à l'équipage mobile aux deux pages précédentes.

4. **Les contraintes d'accessibilité : conteneurs et dispositifs de contiguïté et d'enclos**

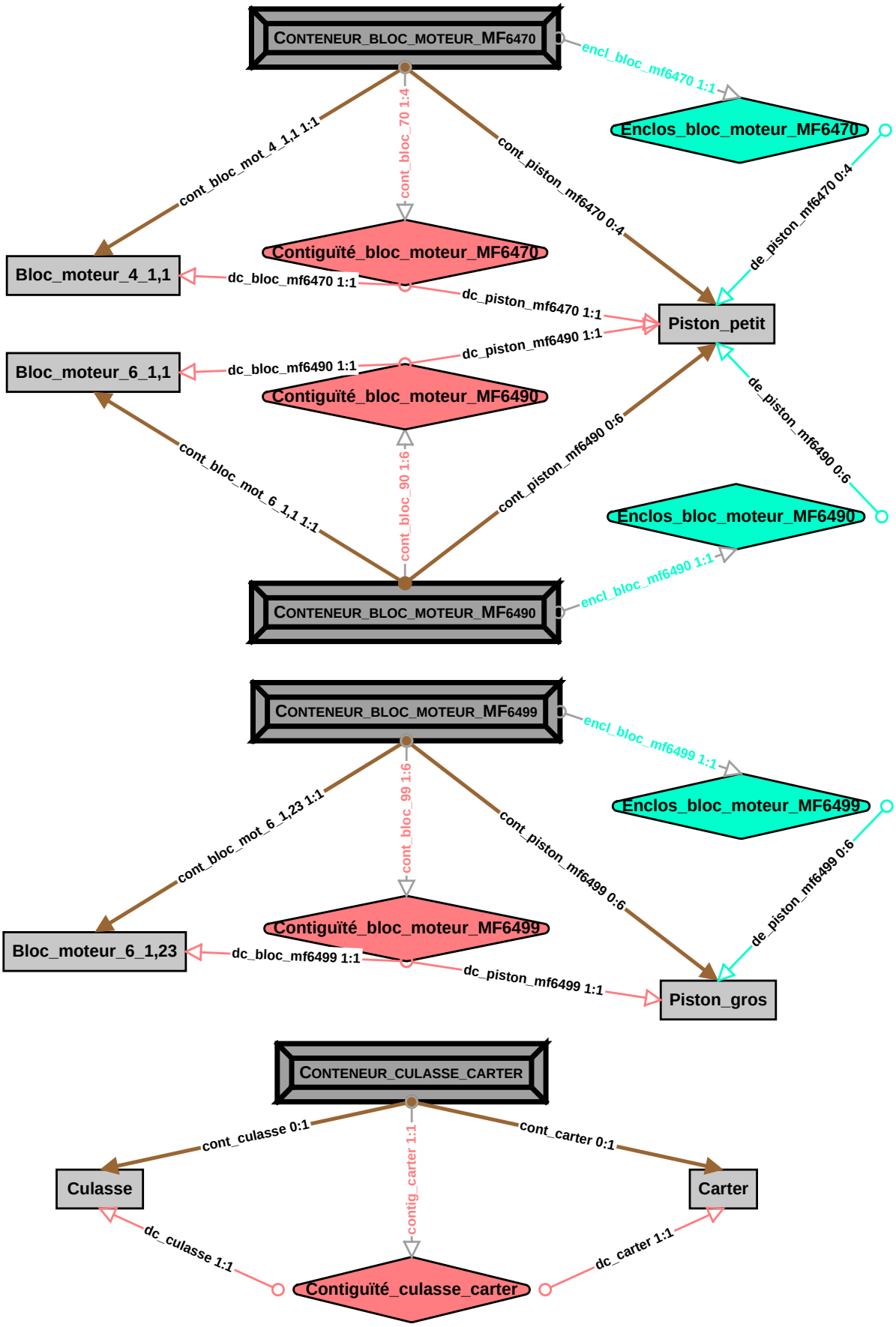


Figure D.14 : Contraintes d'accessibilité de la classe de parts composées *Moteur*

5. **Les piles et les contraintes d'accessibilité**

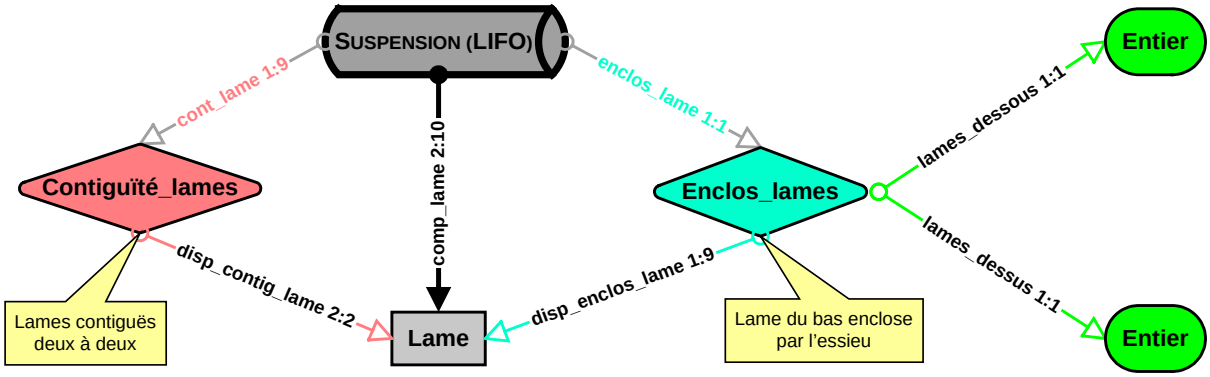


Figure D.15 : Contraintes d'accessibilité de la classe de piles *Suspension*

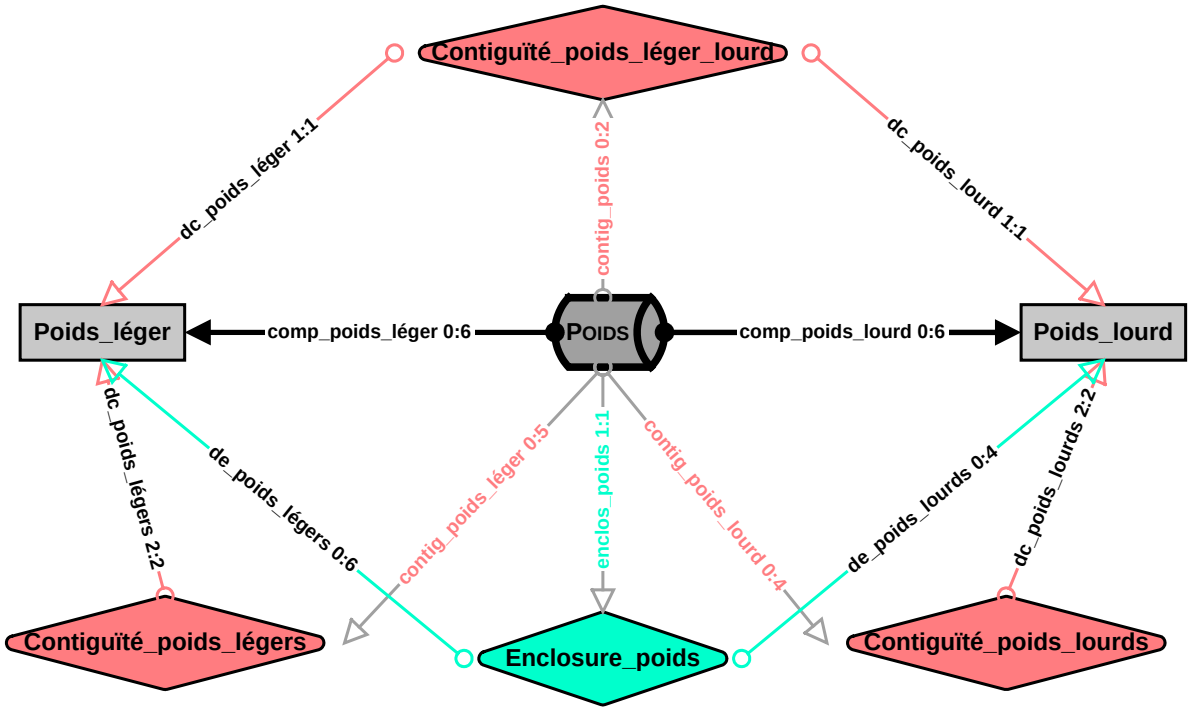


Figure D.16 : Contraintes d'accessibilité de la classe de piles *Poids*

6. Les dispositifs géométriques et de fixation

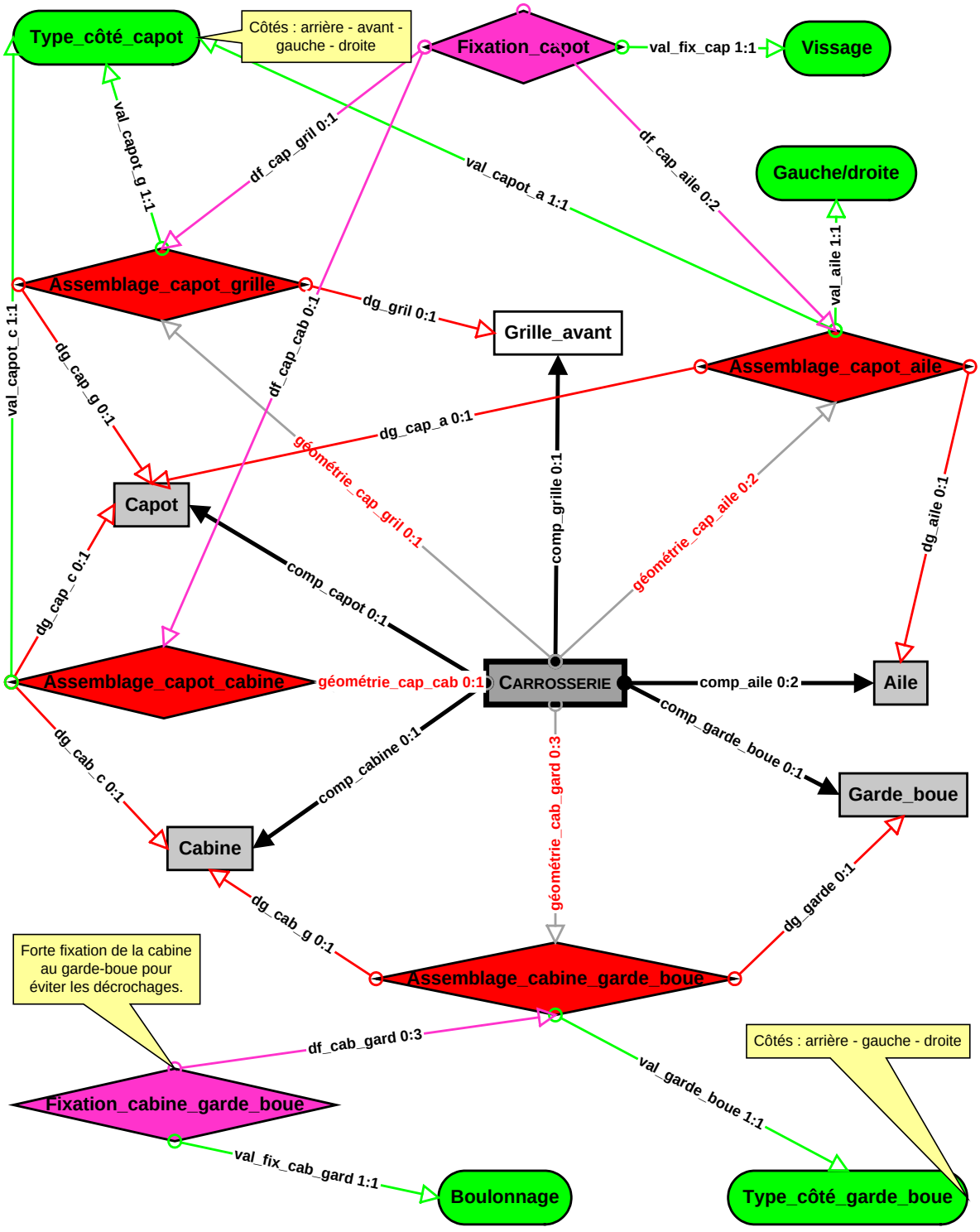


Figure D.17 : Classes de dispositifs géométriques et de fixation
de la classe de parts composées *Carrosserie*

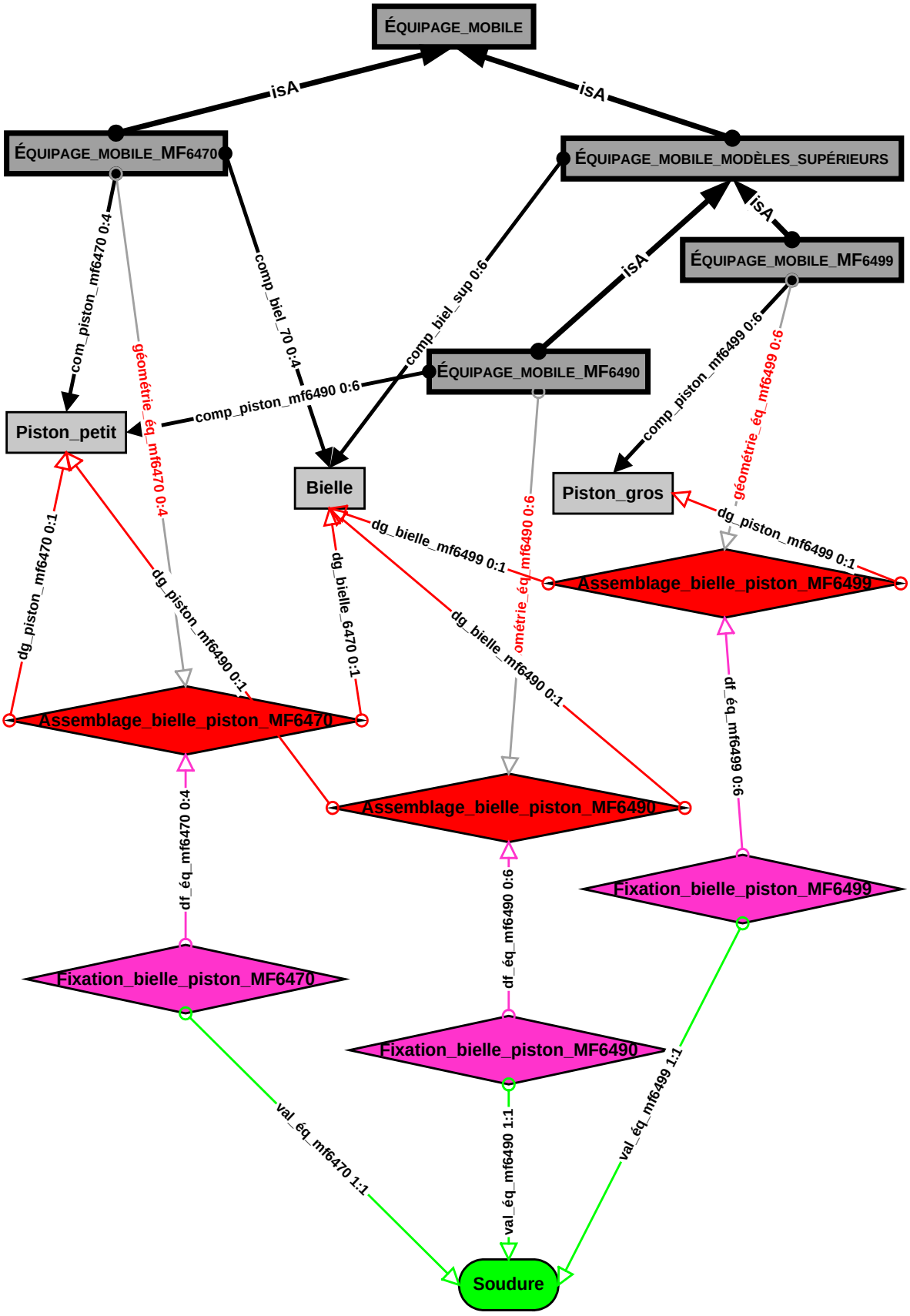


Figure D.18 : Classes de dispositifs géométriques et de fixation
de la classe de parts composées *Équipage_mobile*

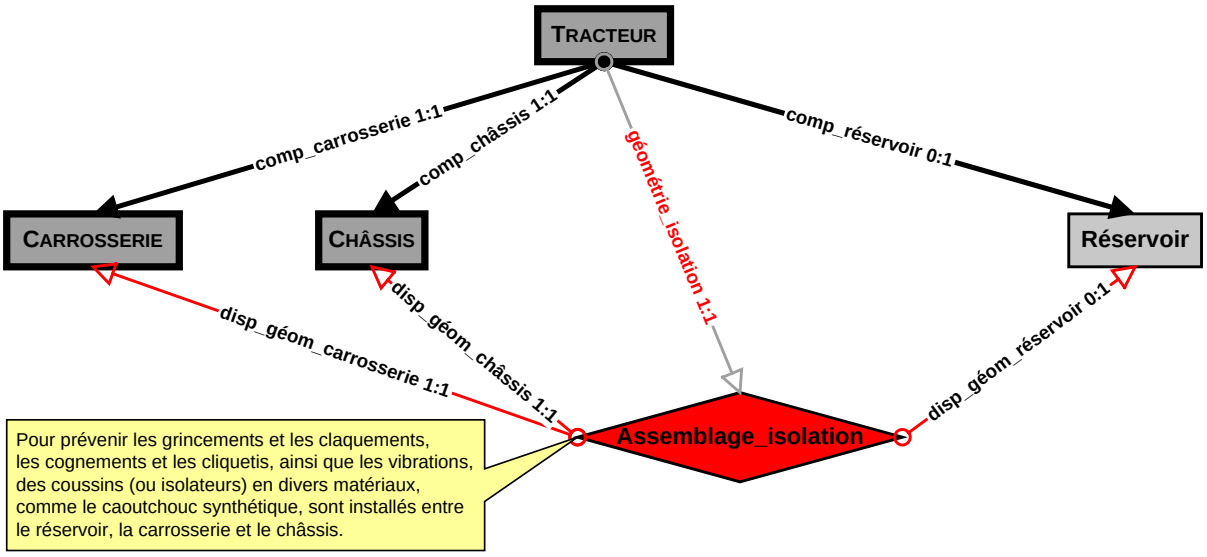


Figure D.19 : Classe de dispositifs géométriques
de la classe de parts composées *Tracteur*

7. **Les dispositifs physiques (et de valeur)**

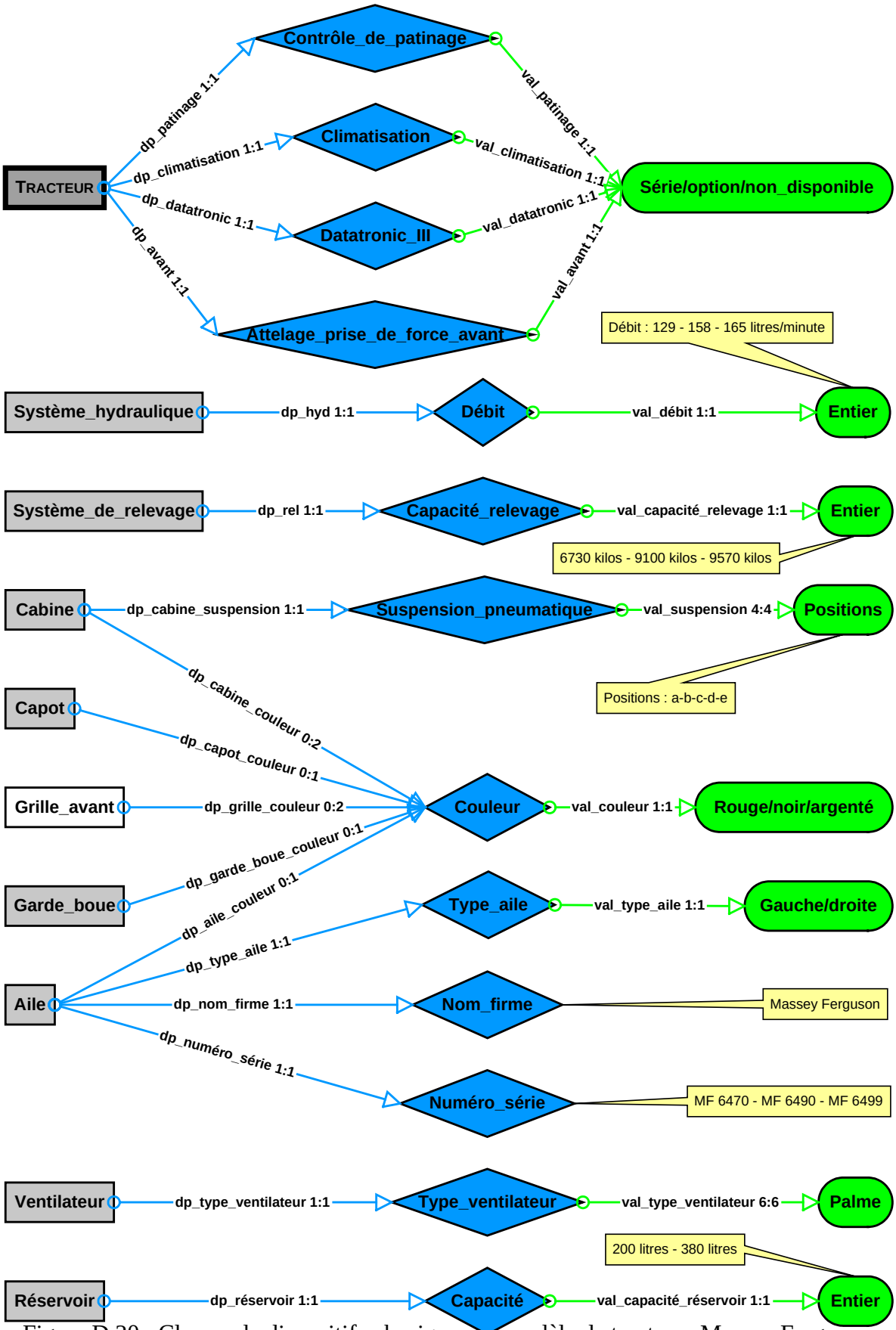


Figure D.20 : Classes de dispositifs physiques du modèle de tracteurs Massey Ferguson

D.4. Modèle exprimé en langage AlbertII

```
%SPEC Exemple Massey Ferguson

%BASIC TYPES

Bâtiment

Atelier

Numéro_châssis

Numéro_moteur

Soudure

Type_côté_garde_boue
    //Côtés : arrière - gauche - droite

Type_côté_capot
    //Côtés : arrière - avant - gauche - droite

Gauche/droite

Boulonnage

Vissage

Série/option/non_disponible

Acier

Modèle_tracteur

Rouge/noir/argenté

Positions
    //Positions : a-b-c-d-e

Nom_firme
    //Massey Ferguson

Numéro_série
    //MF 6470 - MF 6490 - MF 6499

Palme


%CONSTRUCTED TYPES

//CLASSES DE PARTS ÉLÉMENTAIRES

Tracteur = CP[type:CompoundPart,
     id_tracteur1:INTEGER,
     id_tracteur2:STRING,
```

```
      pos_tracteur1:Bâtiment,
      pos_tracteur2:Atelier,
      comp_châssis:Châssis,
      comp_carrosserie:Carrosserie,
      comp_poids:SET[Poids],
      comp_réservoir:Réservoir*,
      géométrie_isolation:Assemblage_isolation,
      dp_patinage:Contrôle_de_patinage,
      dp_climatisation:Climatisation,
      dp_datatronic:Datatronic_III,
      dp_avant:Attelage_prise_de_force_avant]
    //Contrainte d'identité
    \WITH \ForAll p/Tracteur \and \ForAll q/Tracteur : p <> q =>
          id_tracteur1(p) <> id_tracteur1(q) \or
          id_tracteur2(p) <> id_tracteur2(q)
    //Contrainte de cardinalité
    \WITH \ForAll p/Tracteur : Card(comp_poids) <= 12

Châssis = CP[type:CompoundPart,
      id_châssis:Numéro_châssis,
      comp_train_roul:Train_de_roulement,
      comp_moteur:Moteur*,
      comp_transmission:Transmission*,
      comp_cadre:Cadre*,
      comp_éq_élec:Équipement_électrique*,
      comp_prise_force:Prise_de_force*,
      comp_sys_hyd:Système_hydraulique*,
      comp_sys_rel:Système_de_relevage*]
    //Contrainte d'identité
    \WITH \ForAll p/Châssis \and \ForAll q/Châssis : p <> q =>
          id_châssis(p) <> id_châssis(q)
    //Contrainte de composition
    \WITH \ForAll p/Châssis : Card(comp_sys_rel) = 1 \or
          Card(comp_sys_hyd) = 1 \or
          Card(comp_prise_force) = 1 \or
          Card(comp_éq_élec) = 1 \or
          Card(comp_cadre) = 1 \or
          Card(comp_transmission) = 1 \or
          Card(comp_moteur) = 1

Cadre = CP[type:BasicPart,
      id_cadre:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Cadre \and \ForAll q/Cadre : p <> q =>
          id_cadre(p) <> id_cadre(q)

Train_de_roulement = CP[type:CompoundPart,
      id_train_roul:INTEGER,
      comp_pont_avant:Pont_avant,
      comp_pont_arrière:Pont_arrière]
    //Contrainte d'identité
    \WITH \ForAll p/Train_de_roulement \and \ForAll
          q/Train_de_roulement : p <> q =>
          id_train_roul(p) <> id_train_roul(q)
```

```
Pont_avant = CP[type:CompoundPart,
      id_pont_avant:INTEGER,
      comp_roue_avant:SET[Roue_avant],
      comp_essieu_pont_ar:Essieu*]
    //Contrainte d'identité
    \WITH \ForAll p/Pont_avant \and \ForAll q/Pont_avant : p <> q =>
          id_pont_avant(p) <> id_pont_avant(q)
    //Contrainte de composition
    \WITH \ForAll p/Pont_avant : (Card(comp_essieu_pont_ar) +
          Card(comp_roue_avant)) >= 2
    //Contrainte de cardinalité
    \WITH \ForAll p/Pont_avant : Card(comp_roue_avant) <= 2

Pont_arrière = CP[type:CompoundPart,
      id_pont_arrière:INTEGER,
      comp_roue_arrière:SET[Roue_arrière],
      comp_essieu_pont_av:Essieu*,
      comp_suspension:Suspension*]
    //Contrainte d'identité
    \WITH \ForAll p/Pont_arrière \and \ForAll q/Pont_arrière : p <>
          q => id_pont_arrière(p) <> id_pont_arrière(q)
    //Contrainte de composition
    \WITH \ForAll p/Pont_arrière : (Card(comp_suspension) +
          Card(comp_essieu_pont_av) + Card(comp_roue_arrière)) >= 2
    //Contrainte de cardinalité
    \WITH \ForAll p/Pont_arrière : Card(comp_roue_arrière) <= 2

Suspension = CP[type:PileOfParts,
      id_suspension:INTEGER,
      comp_lame:SET[Lame],
      cont_lame:SET[Contiguïté_lames],
      enclos_lame:Enclos_lames]
    //Contrainte d'identité
    \WITH \ForAll p/Suspension \and \ForAll q/Suspension : p <> q =>
          id_suspension(p) <> id_suspension(q)
    //Contrainte de cardinalité
    \WITH \ForAll p/Suspension : Card(cont_lame) >= 1
         \and Card(cont_lame) <= 9
         \and Card(comp_lame) >= 2
         \and Card(comp_lame) <= 10

Lame = CP[type:BasicPart,
      id_lame:INTEGER,
      dp_lame:Acier]
    //Contrainte d'identité
    \WITH \ForAll p/Lame \and \ForAll q/Lame : p <> q =>
          id_lame(p) <> id_lame(q)

Moteur = CP[type:CompoundPart,
      id_moteur:Numéro_moteur,
      comp_éq_fixe:Équipage_fixe,
      comp_éq_mobile:Équipage_mobile,
      comp_accessoires:Accessoires*]
    //Contrainte d'identité
    \WITH \ForAll p/Moteur \and \ForAll q/Moteur : p <> q =>
          id_moteur(p) <> id_moteur(q)
```

```
    //Moteur diesel 4 temps à injection
    //MF 6470 de 125 chevaux avec un moteur 4 cylindres de 4,4
litres
    //MF 6490 de 170 chevaux avec un moteur 6 cylindres de 6,6
litres
    //MF 6499 de 215 chevaux avec un moteur 6 cylindres de 7,4
litres

Équipage_fixe = CP[type:CompoundPart,
     id_éq_fixe:INTEGER,
     comp_bloc_moteur:Bloc_moteur,
     comp_carter:Carter*,
     comp_culasse:Culasse*]
    //Contrainte d'identité
    \WITH \ForAll p/Équipage_fixe \and \ForAll q/Équipage_fixe :
          p <> q => id_éq_fixe(p) <> id_éq_fixe(q)
    //Contrainte de composition
    \WITH \ForAll p/Équipage_fixe : Card(comp_culasse) = 1 \or
          Card(comp_carter) = 1

Culasse = CP[type:BasicPart,
     id_culasse:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Culasse \and \ForAll q/Culasse : p <> q =>
          id_culasse(p) <> id_culasse(q)

Carter = CP[type:BasicPart,
     id_carter:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Carter \and \ForAll q/Carter : p <> q =>
          id_carter(p) <> id_carter(q)

Bielle = CP[type:BasicPart,
     id_bielle:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Bielle \and \ForAll q/Bielle : p <> q =>
          id_bielle(p) <> id_bielle(q)

Piston_petit = CP[type:BasicPart,
     id_piston_petit:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Piston_petit \and \ForAll q/Piston_petit :
          p <> q => id_piston_petit(p) <> id_piston_petit(q)

Piston_gros = CP[type:BasicPart,
     id_piston_gros:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Piston_gros \and \ForAll q/Piston_gros : p <> q =>
          id_piston_gros(p) <> id_piston_gros(q)

Vilebrequin = CP[type:BasicPart,
     id_vilebrequin:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Vilebrequin \and \ForAll q/Vilebrequin : p <> q =>
          id_vilebrequin(p) <> id_vilebrequin(q)
```

```
Segments = CP[type:BasicPart,
      id_segments:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Segments \and \ForAll q/Segments : p <> q =>
          id_segments(p) <> id_segments(q)

Distribution = CP[type:BasicPart,
      id_distribution:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Distribution \and \ForAll q/Distribution : p <> q =>
          id_distribution(p) <> id_distribution(q)

Essieu = CP[type:BasicPart,
      id_essieu:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Essieu \and \ForAll q/Essieu : p <> q =>
          id_essieu(p) <> id_essieu(q)

Roue_avant = CP[type:BasicPart,
      id_roue_avant:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Roue_avant \and \ForAll q/Roue_avant : p <> q =>
          id_roue_avant(p) <> id_roue_avant(q)

Roue_arrière = CP[type:BasicPart,
      id_roue_arrière:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Roue_arrière \and \ForAll q/Roue_arrière : p <> q =>
          id_roue_arrière(p) <> id_roue_arrière(q)

Conteneur_bloc_moteur_MF6470 = CP[type:Container,
      id_cont_bl_mot_MF6470:INTEGER,
      cont_piston_MF6470:SET[Piston_petit],
      cont_bloc_mot_4_1,1:Bloc_moteur_4_1,1,
      cont_bloc_70:SET[Contiguïté_bloc_moteur_MF6470],
      encl_bloc_MF6470:Enclos_bloc_moteur_MF6470]
    //Contrainte d'identité
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6470 \and \ForAll
          q/Conteneur_bloc_moteur_MF6470 : p <> q =>
          id_cont_bl_mot_MF6470(p) <> id_cont_bl_mot_MF6470(q)
    //Contrainte de contenance
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6470 :
          Card(cont_piston_MF6470) >= 1
    //Contrainte de cardinalité
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6470 :
          Card(cont_bloc_70) >= 1
         \and Card(cont_bloc_70) <= 4
         \and Card(cont_piston_MF6470) <= 4

Conteneur_bloc_moteur_MF6490 = CP[type:Container,
      id_cont_bl_mot_MF6490:INTEGER,
      cont_bloc_90:SET[Contiguïté_bloc_moteur_MF6490],
      cont_piston_MF6490:SET[Piston_petit],
      cont_bloc_mot_6_1,1:Bloc_moteur_6_1,1,
      encl_bloc_MF6490:Enclos_bloc_moteur_MF6490]
    //Contrainte d'identité
```

```
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6490 \and \ForAll
          q/Conteneur_bloc_moteur_MF6490 : p <> q =>
          id_cont_bl_mot_MF6490(p) <> id_cont_bl_mot_MF6490(q)
    //Contrainte de contenance
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6490 :
          Card(cont_piston_MF6490) >= 1
    //Contrainte de cardinalité
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6490 :
          Card(cont_piston_MF6490) <= 6
         \and Card(cont_bloc_90) >= 1
         \and Card(cont_bloc_90) <= 6

Conteneur_bloc_moteur_MF6499 = CP[type:Container,
     id_cont_bl_mot_MF6499:INTEGER,
     cont_piston_MF6499:SET[Piston_gros],
     cont_bloc_mot_6_1,23:Bloc_moteur_6_1,23,
     cont_bloc_99:SET[Contiguïté_bloc_moteur_MF6499],
     encl_bloc_MF6499:Enclos_bloc_moteur_MF6499]
    //Contrainte d'identité
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6499 \and \ForAll
          q/Conteneur_bloc_moteur_MF6499 : p <> q =>
          id_cont_bl_mot_MF6499(p) <> id_cont_bl_mot_MF6499(q)
    //Contrainte de contenance
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6499 :
          Card(cont_piston_MF6499) >= 1
    //Contrainte de cardinalité
    \WITH \ForAll p/Conteneur_bloc_moteur_MF6499 :
          Card(cont_bloc_99) >= 1
         \and Card(cont_bloc_99) <= 6
         \and Card(cont_piston_MF6499) <= 6

Accessoires = CP[type:CompoundPart,
     id_accessoires:INTEGER,
     comp_pompe_huile:Pompe_huile*,
     comp_pompe_eau:Pompe_eau*,
     comp_démarreur:Démarreur*,
     comp_alternateur:Alternateur*]
    //Contrainte d'identité
    \WITH \ForAll p/Accessoires \and \ForAll q/Accessoires : p <> q =>
          id_accessoires(p) <> id_accessoires(q)
    //Contrainte de composition
    \WITH \ForAll p/Accessoires : (Card(comp_alternateur) +
          Card(comp_démarreur) + Card(comp_pompe_eau) +
          Card(comp_pompe_huile)) >= 2

Pompe_eau = CP[type:BasicPart,
     id_pompe_eau:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Pompe_eau \and \ForAll q/Pompe_eau : p <> q =>
          id_pompe_eau(p) <> id_pompe_eau(q)

Pompe_huile = CP[type:BasicPart,
     id_pompe_huile:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Pompe_huile \and \ForAll q/Pompe_huile : p <> q =>
          id_pompe_huile(p) <> id_pompe_huile(q)
```

```
Alternateur = CP[type:BasicPart,
      id_alternateur:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Alternateur \and \ForAll q/Alternateur : p <> q =>
          id_alternateur(p) <> id_alternateur(q)

Démarreur = CP[type:BasicPart,
      id_démarreur:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Démarreur \and \ForAll q/Démarreur : p <> q =>
          id_démarreur(p) <> id_démarreur(q)

Conteneur_culasse_carter = CP[type:Container,
      id_Conteneur_culasse_carter:INTEGER,
      cont_carter:Carter*,
      cont_culasse:Culasse*,
      contig_carter:Contiguïté_culasse_carter]
    //Contrainte d'identité
    \WITH \ForAll p/Conteneur_culasse_carter \and \ForAll
          q/Conteneur_culasse_carter : p <> q =>
          id_Conteneur_culasse_carter(p) <>
          id_Conteneur_culasse_carter(q)
    //Contrainte de contenance
    \WITH \ForAll p/Conteneur_culasse_carter : (Card(cont_culasse) +
          Card(cont_carter)) >= 2

Transmission = CP[type:BasicPart,
      id_transmission:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Transmission \and \ForAll q/Transmission : p <> q =>
          id_transmission(p) <> id_transmission(q)

Équipement_électrique = CP[type:BasicPart,
      id_éq_élec:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Équipement_électrique \and \ForAll
          q/Équipement_électrique : p <> q =>
          id_éq_élec(p) <> id_éq_élec(q)

Prise_de_force = CP[type:BasicPart,
      id_prise_force:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Prise_de_force \and \ForAll q/Prise_de_force :
          p <> q => id_prise_force(p) <> id_prise_force(q)

Système_hydraulique = CP[type:BasicPart,
      id_sys_hyd:INTEGER,
      dp_hyd:Débit]
    //Contrainte d'identité
    \WITH \ForAll p/Système_hydraulique \and \ForAll
          q/Système_hydraulique : p <> q =>
          id_sys_hyd(p) <> id_sys_hyd(q)

Système_de_relevage = CP[type:BasicPart,
      id_sys_rel:INTEGER,
      dp_rel:Capacité_relevage]
```

```
    //Contrainte d'identité
    \WITH \ForAll p/Système_de_relevage \and \ForAll
          q/Système_de_relevage : p <> q =>
          id_sys_rel(p) <> id_sys_rel(q)

Carrosserie = CP[type:CompoundPart,
     id_carrosserie:INTEGER,
     comp_capot:Capot*,
     comp_cabine:Cabine*,
     comp_aile:SET[Aile],
     comp_garde_boue:Garde_boue*,
     comp_grille:Grille_avant*,
     géométrie_cab_gard:SET[Assemblage_cabine_garde_boue],
     géométrie_cap_cab:Assemblage_capot_cabine*,
     géométrie_cap_gril:Assemblage_capot_grille*,
     géométrie_cap_aile:SET[Assemblage_capot_aile]]
    //Contrainte d'identité
    \WITH \ForAll p/Carrosserie \and \ForAll q/Carrosserie : p <> q =>
          id_carrosserie(p) <> id_carrosserie(q)
    //Contrainte de composition
    \WITH \ForAll p/Carrosserie : (Card(comp_grille) +
          Card(comp_garde_boue) + Card(comp_aile) +
          Card(comp_cabine) + Card(comp_capot)) >= 2
    //Contrainte de cardinalité
    \WITH \ForAll p/Carrosserie : Card(géométrie_cap_aile) <= 2
         \and Card(géométrie_cab_gard) <= 3
         \and Card(comp_aile) <= 2

Cabine = CP[type:BasicPart,
     id_cabine:INTEGER,
     dp_cabine_couleur:SET[Couleur],
     dp_cabine_suspension:Suspension_pneumatique]
    //Contrainte d'identité
    \WITH \ForAll p/Cabine \and \ForAll q/Cabine : p <> q =>
          id_cabine(p) <> id_cabine(q)
    //Contrainte de cardinalité
    \WITH \ForAll p/Cabine : Card(dp_cabine_couleur) <= 2

Capot = CP[type:BasicPart,
     id_capot:INTEGER,
     dp_capot_couleur:Couleur*]
    //Contrainte d'identité
    \WITH \ForAll p/Capot \and \ForAll q/Capot : p <> q =>
          id_capot(p) <> id_capot(q)

Aile = CP[type:BasicPart,
     id_aile:INTEGER,
     dp_aile_couleur:Couleur*,
     dp_type_aile:Type_aile,
     dp_nom_firme:Nom_firme,
     dp_numéro_série:Numéro_série]
    //Contrainte d'identité
    \WITH \ForAll p/Aile \and \ForAll q/Aile : p <> q =>
          id_aile(p) <> id_aile(q)
```

```
Grille_grosse = CP[type:BasicPart,
      id_grille_grosse:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Grille_grosse \and \ForAll q/Grille_grosse :
          p <> q => id_grille_grosse(p) <> id_grille_grosse(q)

Ventilateur = CP[type:BasicPart,
      id_ventilateur:INTEGER,
      dp_type_ventilateur:Type_ventilateur]
    //Contrainte d'identité
    \WITH \ForAll p/Ventilateur \and \ForAll q/Ventilateur : p <> q =>
          id_ventilateur(p) <> id_ventilateur(q)

Garde_boue = CP[type:BasicPart,
      id_garde_boue:INTEGER,
      dp_garde_boue_couleur:Couleur*]
    //Contrainte d'identité
    \WITH \ForAll p/Garde_boue \and \ForAll q/Garde_boue : p <> q =>
          id_garde_boue(p) <> id_garde_boue(q)

Réservoir = CP[type:BasicPart,
      id_réservoir:INTEGER,
      dp_réservoir:Capacité]
    //Contrainte d'identité
    \WITH \ForAll p/Réservoir \and \ForAll q/Réservoir : p <> q =>
          id_réservoir(p) <> id_réservoir(q)

Poids = CP[type:PileOfParts,
      id_poids:INTEGER,
      comp_poids_léger:SET[Poids_léger],
      comp_poids_lourd:SET[Poids_lourd],
      contig_poids_léger:SET[Contiguïté_poids_légers],
      contig_poids_lourd:SET[Contiguïté_poids_lourds],
      contig_poids:SET[Contiguïté_poids_léger_lourd],
      enclos_poids:Enclosure_poids]
    //Contrainte d'identité
    \WITH \ForAll p/Poids \and \ForAll q/Poids : p <> q =>
          id_poids(p) <> id_poids(q)
    //Contrainte de composition
    \WITH \ForAll p/Poids : (Card(comp_poids_lourd) +
          Card(comp_poids_léger)) >= 2
    //Contrainte de cardinalité
    \WITH \ForAll p/Poids : Card(contig_poids) <= 2
         \and Card(contig_poids_lourd) <= 4
         \and Card(contig_poids_léger) <= 5
         \and Card(comp_poids_lourd) <= 6
         \and Card(comp_poids_léger) <= 6
    //Contrainte de contiguïté
    \WITH \ForAll p/Poids : Card(contig_poids) >= 1 \or
          Card(contig_poids_lourd) >= 1 \or
          Card(contig_poids_léger) >= 1
    //Promotion : offre gratuite d’un jeu de 12 poids

Poids_léger = CP[type:BasicPart,
      id_poids_léger:INTEGER]
    //Contrainte d'identité
```

```
    \WITH \ForAll p/Poids_léger \and \ForAll q/Poids_léger : p <> q =>
          id_poids_léger(p) <> id_poids_léger(q)

Poids_lourd = CP[type:BasicPart,
     id_poids_lourd:INTEGER]
    //Contrainte d'identité
    \WITH \ForAll p/Poids_lourd \and \ForAll q/Poids_lourd : p <> q =>
          id_poids_lourd(p) <> id_poids_lourd(q)


//SUPER CLASSES DE PARTS RACINES

Bloc_moteur = CP[id_bloc_moteur:INTEGER,
     type:ENUM[Bloc_moteur_4_1,1,
          Bloc_moteur_6_1,23,
          Bloc_moteur_6_1,1],
     dp_bloc_moteur:Modèle_tracteur]
    //Contrainte d'identité
    \WITH \ForAll p/Bloc_moteur \and \ForAll q/Bloc_moteur : p <> q =>
          id_bloc_moteur(p) <> id_bloc_moteur(q)
    //Contrainte d'énumération
    \WITH \ForAll p : Bloc_moteur
         (Type(p)=Bloc_moteur_4_1,1) \and
         (Type(p)=Bloc_moteur_6_1,23) \and
         (Type(p)=Bloc_moteur_6_1,1)

Équipage_mobile = CP[id_éq_mobile:INTEGER,
     comp_vilebrequin:Vilebrequin*,
     comp_segments:Segments*,
     com_distribution:Distribution*,
     type:ENUM[Équipage_mobile_MF6470,
          Équipage_mobile_MF6490,
          Équipage_mobile_MF6499],
     comp_biel_70:SET[Bielle],
     com_piston_MF6470:SET[Piston_petit],
     géométrie_éq_MF6470:SET[Assemblage_bielle_piston_MF6470],
     comp_biel_sup:SET[Bielle],
     comp_piston_MF6490:SET[Piston_petit],
     géométrie_éq_MF6490:SET[Assemblage_bielle_piston_MF6490],
     comp_piston_MF6499:SET[Piston_gros],
     géométrie_éq_MF6499:SET[Assemblage_bielle_piston_MF6499]]
    //Contrainte d'identité
    \WITH \ForAll p/Équipage_mobile \and \ForAll q/Équipage_mobile :
          p <> q => id_éq_mobile(p) <> id_éq_mobile(q)
    //Contrainte d'énumération
    \WITH \ForAll p : Équipage_mobile
         (Type(p)=Équipage_mobile_MF6470 <=>
          comp_biel_70 <> \undef \and
          com_piston_MF6470 <> \undef \and
          géométrie_éq_MF6470 <> \undef \and
          comp_piston_MF6490 = \undef \and
          géométrie_éq_MF6490 = \undef \and
          comp_piston_MF6499 = \undef \and
          géométrie_éq_MF6499 = \undef \and
          comp_biel_sup = \undef \and
           (Card(com_piston_MF6470) + Card(comp_biel_70) +
```

```
            Card(com_distribution) + Card(comp_segments) +
            Card(comp_vilebrequin)) >= 2 \and
            Card(géométrie_éq_MF6470) <= 4 \and
            Card(com_piston_MF6470) <= 4 \and
            Card(comp_biel_70) <= 4) \and
           (Type(p)=Équipage_mobile_MF6490 <=>
            comp_piston_MF6490 <> \undef \and
            géométrie_éq_MF6490 <> \undef \and
            comp_biel_sup <> \undef \and
            comp_biel_70 = \undef \and
            com_piston_MF6470 = \undef \and
            géométrie_éq_MF6470 = \undef \and
            comp_piston_MF6499 = \undef \and
            géométrie_éq_MF6499 = \undef \and
            (Card(comp_piston_MF6490) + Card(comp_biel_sup) +
            Card(com_distribution) + Card(comp_segments) +
            Card(comp_vilebrequin)) >= 2 \and
            Card(géométrie_éq_MF6490) <= 6 \and
            Card(comp_piston_MF6490) <= 6 \and
            Card(comp_biel_sup) <= 6) \and
           (Type(p)=Équipage_mobile_MF6499 <=>
            comp_piston_MF6499 <> \undef \and
            géométrie_éq_MF6499 <> \undef \and
            comp_biel_sup <> \undef \and
            comp_biel_70 = \undef \and
            com_piston_MF6470 = \undef \and
            géométrie_éq_MF6470 = \undef \and
            comp_piston_MF6490 = \undef \and
            géométrie_éq_MF6490 = \undef \and
            (Card(comp_piston_MF6499) + Card(comp_biel_sup) +
            Card(com_distribution) + Card(comp_segments) +
            Card(comp_vilebrequin)) >= 2 \and
            Card(géométrie_éq_MF6499) <= 6 \and
            Card(comp_piston_MF6499) <= 6 \and
            Card(comp_biel_sup) <= 6)

Grille_avant = CP[id_grille_avant:INTEGER,
     type:ENUM[Grille_simple, Grille_composée],
     comp_ventilateur:Ventilateur*,
     comp_grille_grosse:Grille_grosse*,
     dp_grille_couleur:SET[Couleur]]
    //Contrainte d'identité
    \WITH \ForAll p/Grille_avant \and \ForAll q/Grille_avant : p <> q =>
          id_grille_avant(p) <> id_grille_avant(q)
    //Contrainte d'énumération
    \WITH \ForAll p : Grille_avant
         (Type(p)=Grille_simple <=>
          comp_ventilateur = \undef \and
          comp_grille_grosse = \undef \and
          Card(dp_grille_couleur) <= 2) \and
         (Type(p)=Grille_composée <=>
          comp_ventilateur <> \undef \and
          comp_grille_grosse <> \undef \and
          Card(dp_grille_couleur) <= 2)
```

```
//CLASSES DE PARTS SOUS-ENSEMBLES ET SUPER CLASSES

Équipage_mobile_modèles_supérieurs = Équipage_mobile


//SOUS-CLASSES DE PARTS FEUILLES

Bloc_moteur_4_1,1 = CP[type:BasicPart,
     isA:Bloc_moteur]
    //125 chevaux

Bloc_moteur_6_1,1 = CP[type:BasicPart,
     isA:Bloc_moteur]
    //170 chevaux

Bloc_moteur_6_1,23 = CP[type:BasicPart,
     isA:Bloc_moteur]
    //215 chevaux

Équipage_mobile_MF6470 = CP[type:CompoundPart,
     isA:Équipage_mobile]

Équipage_mobile_MF6490 = CP[type:CompoundPart,
     isA:Équipage_mobile_modèles_supérieurs]

Équipage_mobile_MF6499 = CP[type:CompoundPart,
     isA:Équipage_mobile_modèles_supérieurs]

Grille_simple = CP[type:BasicPart,
     isA:Grille_avant]

Grille_composée = CP[type:CompoundPart,
     isA:Grille_avant]


//CLASSES DE DISPOSITIFS PHYSIQUES

Climatisation = CP[type:PartPhysicalFeature,
     val_climatisation:Série/option/non_disponible]

Contrôle_de_patinage = CP[type:PartPhysicalFeature,
     val_patinage:Série/option/non_disponible]

Datatronic_III = CP[type:PartPhysicalFeature,
     val_datatronic:Série/option/non_disponible]

Attelage_prise_de_force_avant = CP[type:PartPhysicalFeature,
     val_avant:Série/option/non_disponible]

Débit = CP[type:PartPhysicalFeature,
     val_débit:INTEGER]
     //Débit : 129 - 158 - 165 litres/minute

Capacité_relevage = CP[type:PartPhysicalFeature,
     val_capacité_relevage:INTEGER]
     //6730 kilos - 9100 kilos - 9570 kilos
```

```
Couleur = CP[type:PartPhysicalFeature,
     val_couleur:Rouge/noir/argenté]

Suspension_pneumatique = CP[type:PartPhysicalFeature,
     val_suspension:BAG[Positions]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Suspension_pneumatique : Card(val_suspension) = 4

Type_aile = CP[type:PartPhysicalFeature,
     val_type_aile:Gauche/droite]

Type_ventilateur = CP[type:PartPhysicalFeature,
     val_type_ventilateur:SEQ[Palme]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Type_ventilateur : Card(val_type_ventilateur) = 6

Capacité = CP[type:PartPhysicalFeature,
     val_capacité_réservoir:INTEGER]
     //200 litres - 380 litres

//CLASSES DE DISPOSITIFS GÉOMÉTRIQUES

Assemblage_isolation = CP[type:PartGeometricalFeature,
     disp_géom_carrosserie:Carrosserie,
     disp_géom_châssis:Châssis,
     disp_géom_réservoir:Réservoir*]
    //Pour prévenir les grincements et les claquements, les
cognements et les cliquetis, ainsi que les vibrations, des coussins
(ou isolateurs) en divers matériaux, comme le caoutchouc
synthétique, sont installés entre le réservoir, la carrosserie et le
châssis.

Assemblage_bielle_piston_MF6470 = CP[type:PartGeometricalFeature,
     dg_piston_MF6470:Piston_petit*,
     dg_bielle_6470:Bielle*]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_bielle_piston_MF6470 :
           (Card(dg_bielle_6470) + Card(dg_piston_MF6470)) >= 2

Assemblage_bielle_piston_MF6490 = CP[type:PartGeometricalFeature,
     dg_piston_MF6490:Piston_petit*,
     dg_bielle_MF6490:Bielle*]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_bielle_piston_MF6490 :
           (Card(dg_bielle_MF6490) + Card(dg_piston_MF6490)) >= 2

Assemblage_bielle_piston_MF6499 = CP[type:PartGeometricalFeature,
     dg_bielle_MF6499:Bielle*,
     dg_piston_MF6499:Piston_gros*]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_bielle_piston_MF6499 :
           (Card(dg_piston_MF6499) + Card(dg_bielle_MF6499)) >= 2

Assemblage_cabine_garde_boue = CP[type:PartGeometricalFeature,
     dg_cab_g:Cabine*,
     dg_garde:Garde_boue*,
```

```
      val_garde_boue:Type_côté_garde_boue]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_cabine_garde_boue :
          (Card(dg_garde) + Card(dg_cab_g)) >= 2

Assemblage_capot_cabine = CP[type:PartGeometricalFeature,
      dg_cap_c:Capot*,
      dg_cab_c:Cabine*,
      val_capot_c:Type_côté_capot]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_capot_cabine :
          (Card(dg_cab_c) + Card(dg_cap_c)) >= 2

Assemblage_capot_grille = CP[type:PartGeometricalFeature,
      dg_cap_g:Capot*,
      dg_gril:Grille_avant*,
      val_capot_g:Type_côté_capot]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_capot_grille :
          (Card(dg_gril) + Card(dg_cap_g)) >= 2

Assemblage_capot_aile = CP[type:PartGeometricalFeature,
      dg_cap_a:Capot*,
      dg_aile:Aile*,
      val_capot_a:Type_côté_capot,
      val_aile:Gauche/droite]
    //Contrainte de géométrie
    \WITH \ForAll p/Assemblage_capot_aile :
          (Card(dg_aile) + Card(dg_cap_a)) >= 2


//CLASSES DE DISPOSITIFS DE FIXATION

Fixation_bielle_piston_MF6470 = CP[type:PartFixingFeature,
      df_éq_MF6470:SET[Assemblage_bielle_piston_MF6470],
      val_éq_MF6470:Soudure]
    //Contrainte de cardinalité
    \WITH \ForAll p/Fixation_bielle_piston_MF6470 :
          Card(df_éq_MF6470) <= 4
    //Contrainte de fixation
    \WITH \ForAll p/Fixation_bielle_piston_MF6470 :
          Card(df_éq_MF6470) >= 1

Fixation_bielle_piston_MF6490 = CP[type:PartFixingFeature,
      df_éq_MF6490:SET[Assemblage_bielle_piston_MF6490],
      val_éq_MF6490:Soudure]
    //Contrainte de cardinalité
    \WITH \ForAll p/Fixation_bielle_piston_MF6490 :
          Card(df_éq_MF6490) <= 6
    //Contrainte de fixation
    \WITH \ForAll p/Fixation_bielle_piston_MF6490 :
          Card(df_éq_MF6490) >= 1

Fixation_bielle_piston_MF6499 = CP[type:PartFixingFeature,
      df_éq_MF6499:SET[Assemblage_bielle_piston_MF6499],
      val_éq_MF6499:Soudure]
```

```
    //Contrainte de cardinalité
    \WITH \ForAll p/Fixation_bielle_piston_MF6499 :
          Card(df_éq_MF6499) <= 6
    //Contrainte de fixation
    \WITH \ForAll p/Fixation_bielle_piston_MF6499 :
          Card(df_éq_MF6499) >= 1

Fixation_cabine_garde_boue = CP[type:PartFixingFeature,
     val_fix_cab_gard:Boulonnage,
     df_cab_gard:SET[Assemblage_cabine_garde_boue]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Fixation_cabine_garde_boue : Card(df_cab_gard) <= 3
    //Contrainte de fixation
    \WITH \ForAll p/Fixation_cabine_garde_boue : Card(df_cab_gard) >= 1
    //Forte fixation de la cabine au garde-boue pour éviter les
décrochages.

Fixation_capot = CP[type:PartFixingFeature,
     val_fix_cap:Vissage,
     df_cap_aile:SET[Assemblage_capot_aile],
     df_cap_gril:Assemblage_capot_grille*,
     df_cap_cab:Assemblage_capot_cabine*]
    //Contrainte de cardinalité
    \WITH \ForAll p/Fixation_capot : Card(df_cap_aile) <= 2
    //Contrainte de fixation
    \WITH \ForAll p/Fixation_capot :
          Card(df_cap_cab) = 1 \or Card(df_cap_gril) = 1 \or
          Card(df_cap_aile) >= 1


//CLASSES DE DISPOSITIFS DE CONTIGUÏTÉ

Contiguïté_bloc_moteur_MF6470 = CP[type:PartContiguityFeature,
     dc_bloc_MF6470:Bloc_moteur_4_1,1,
     dc_piston_MF6470:Piston_petit]

Contiguïté_bloc_moteur_MF6490 = CP[type:PartContiguityFeature,
     dc_bloc_MF6490:Bloc_moteur_6_1,1,
     dc_piston_MF6490:Piston_petit]

Contiguïté_bloc_moteur_MF6499 = CP[type:PartContiguityFeature,
     dc_bloc_MF6499:Bloc_moteur_6_1,23,
     dc_piston_MF6499:Piston_gros]

Contiguïté_culasse_carter = CP[type:PartContiguityFeature,
     dc_culasse:Culasse,
     dc_carter:Carter]

Contiguïté_lames = CP[type:PartContiguityFeature,
     disp_contig_lame:SET[Lame]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Contiguïté_lames : Card(disp_contig_lame) = 2
    //Lames contiguës deux à deux
```

```
Contiguïté_poids_légers = CP[type:PartContiguityFeature,
      dc_poids_légers:SET[Poids_léger]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Contiguïté_poids_légers : Card(dc_poids_légers) = 2

Contiguïté_poids_lourds = CP[type:PartContiguityFeature,
      dc_poids_lourds:SET[Poids_lourd]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Contiguïté_poids_lourds : Card(dc_poids_lourds) = 2

Contiguïté_poids_léger_lourd = CP[type:PartContiguityFeature,
      dc_poids_léger:Poids_léger,
      dc_poids_lourd:Poids_lourd]


//CLASSES DE DISPOSITIFS D'ENCLOS

Enclos_bloc_moteur_MF6470 = CP[type:PartEnclosureFeature,
      de_piston_MF6470:SET[Piston_petit]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Enclos_bloc_moteur_MF6470 :
           Card(de_piston_MF6470) <= 4

Enclos_bloc_moteur_MF6490 = CP[type:PartEnclosureFeature,
      de_piston_MF6490:SET[Piston_petit]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Enclos_bloc_moteur_MF6490 :
           Card(de_piston_MF6490) <= 6

Enclos_bloc_moteur_MF6499 = CP[type:PartEnclosureFeature,
      de_piston_MF6499:SET[Piston_gros]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Enclos_bloc_moteur_MF6499 :
           Card(de_piston_MF6499) <= 6

Enclos_lames = CP[type:PartEnclosureFeature,
      disp_enclos_lame:SET[Lame],
      lames_dessous:INTEGER,
      lames_dessus:INTEGER]
    //Contrainte de cardinalité
    \WITH \ForAll p/Enclos_lames : Card(disp_enclos_lame) >= 1
         \and Card(disp_enclos_lame) <= 9
    //Lame du bas enclose par l'essieu

Enclosure_poids = CP[type:PartEnclosureFeature,
      de_poids_légers:SET[Poids_léger],
      de_poids_lourds:SET[Poids_lourd]]
    //Contrainte de cardinalité
    \WITH \ForAll p/Enclosure_poids : Card(de_poids_lourds) <= 4
         \and Card(de_poids_légers) <= 6
```

Annexe E

Application des règles de mappage sur des fragments de modèles AlbertII et du sous-ensemble du pattern *Part*

Les règles de mappage définies dans la troisième section du dernier chapitre se rapportent aux concepts présents dans les méta-modèles du sous-ensemble du pattern *Part* et du langage AlbertII et établissent toujours une correspondance du premier vers le second. Lorsqu'un modèle particulier est exprimé à l'aide du sous-ensemble, les règles peuvent être appliquées aux instances des concepts présents dans ce modèle. Chaque fois qu'une règle est appliquée, un lien de traçabilité est créé entre les éléments des modèles concernés, éléments qui sont des instances des concepts des langages inclus dans le méta-modèle.

Les figures E.1 et E.2 aux pages suivantes illustrent la définition et l'usage des règles de mappage sur deux fragments de modèles empruntés à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D. Dans le coin supérieur gauche, une partie du méta-modèle du sous-ensemble du Pattern *Part* est représentée. Le fragment de modèle dans le coin inférieur gauche a été créé par instanciation des concepts présents dans ce méta-modèle. Par exemple, la classe de parts mixtes *Grille_avant* est une instance de la méta-classe de parts mixtes (*BasicOrCompoundPartClass*) et la classe d'attributs géométries *géométrie_cap_gril* est une instance de la méta-classe d'attributs géométries (*possible_geometry*). Les contraintes qui sont traduites en contraintes AlbertII proviennent des cardinalités des classes générales qui sont des instances des méta-classes. Les classes générales n'ont pas été reprises dans le schéma pour éviter de l'alourdir.

Le coin supérieur droit présente des concepts du langage AlbertII. Nous préférons employer le terme de "concepts" plutôt que celui de "méta-modèle" comme pour le sous-ensemble parce que les relations entre ces concepts ne sont pas indiquées. Ces relations ne figurent pas non plus dans le méta-modèle AlbertII présenté dans la thèse. [Petit 1999, p. 64] Ce dernier est en effet partiel et le compléter ne fait pas partie des objectifs de notre mémoire. Par conséquent, afin de distinguer les différents types de données construits, nous leur avons accolé un qualificatif emprunté aux classes du sous-ensemble et que l'on retrouve en intitulé (sous forme de commentaires) des différentes parties du code AlbertII généré : élémentaire, racine, feuille et dispositif pour la traduction respective des classes de parts élémentaires, racines et feuilles et des classes de dispositifs physiques et géométriques. Tous les types de données construits sont des produits cartésiens à l'exception de ceux qui traduisent les classes de parts qui sont à la fois des sous-ensembles et des super classes. Mais aucun des deux schémas ne comporte ce genre de classes.

Nous pourrions intuitivement instancier les concepts AlbertII pour obtenir le morceau de modèle dans le coin inférieur droit. Mais les règles de mappage situées au centre des schémas permettent de dériver automatiquement ce modèle du modèle graphique correspondant. Par exemple, la règle 3 indique qu'une méta-classe d'identités (*PartIdentityClass*) du sous-ensemble du pattern *Part* peut être traduite en type de données de base prédéfini AlbertII. La règle 30 stipule qu'une contrainte s'exerçant sur la classe générale d'attributs dispositifs géométriques (contrainte n° 37 : un dispositif géométrique assemble au moins deux parts composantes) peut être traduite en contrainte de géométrie.

Les règles qui transposent le modèle du sous-ensemble en modèle AlbertII sont des instances des règles de mappage citées ci-dessus. Par exemple, la règle 3 appliquée représente l'exécution de la règle 3 sur la classe d'identités *Entier*. Cette exécution donne lieu à la création d'une instance du type de données de base prédéfini AlbertII avec un nom équivalent (en anglais). L'application de l'ensemble des règles permet de dériver de manière systématique le fragment de modèle graphique en un modèle AlbertII correspondant. La règle 32 qui traduit les commentaires est appliquée sans qu'elle soit reprise dans la partie supérieure du schéma car elle ne figure pas dans le méta-modèle du sous-ensemble. Un commentaire peut être associé à n'importe quelle classe de parts ou d'états.

Les règles ne sont pas bidirectionnelles parce qu'elles ne représentent pas une équivalence des concepts. Par exemple, une contrainte de géométrie AlbertII (règle 30) n'est pas directement traduisible en un concept du sous-ensemble parce qu'elle ne s'applique que si la somme des bornes inférieures des cardinalités des classes d'attributs dispositifs géométriques associées à une classe de dispositifs géométriques est inférieure à deux. Autre exemple, le concept AlbertII "`Champ du produit cartésien (règle 8) avec valeurs ordonnées et dupliquées`" (règles 16 et 23) en bas du second schéma permet de construire une classe d'attributs dispositifs de valeur. Mais la cardinalité de cette classe reste indéterminée car elle est définie dans ce cas-ci par la contrainte de cardinalité (règle 27) située juste en dessous.

Les règles de mappage proposées dans le mémoire ont été définies en considérant le sous-ensemble du pattern *Part* comme le langage central puisque les modèles graphiques de "parts" qui permettent d'exprimer les concepts de ce langage doivent être traduits en types de données et contraintes AlbertII. Lorsque le modèle est complet et satisfait aux contraintes vérifiées par l'outil logiciel, celui-ci le traduit automatiquement en un code AlbertII syntaxiquement correct sans que l'utilisateur soit sollicité, notamment dans le choix des sémantiques de concepts à appliquer.

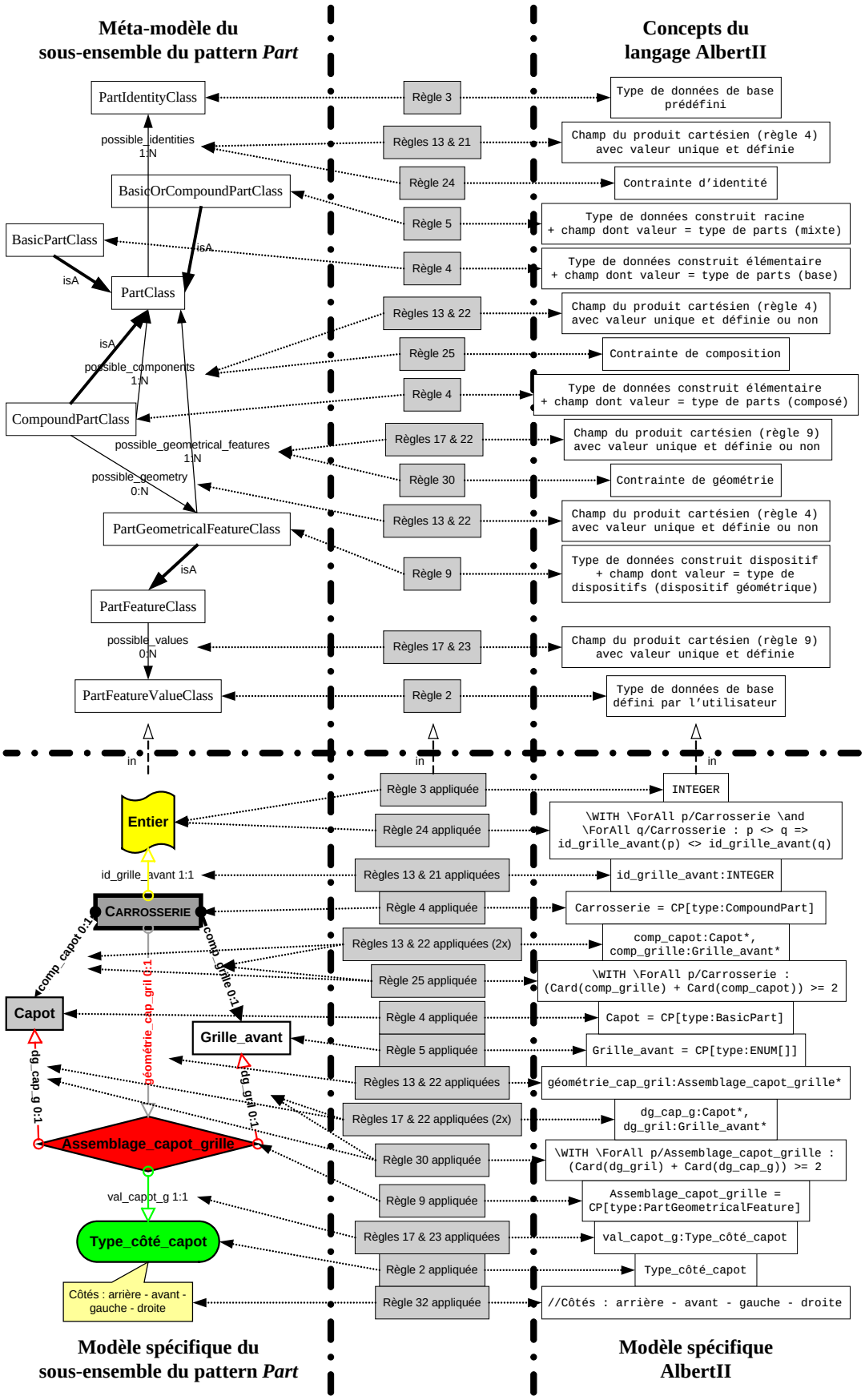


Figure E.1 : Exemple d'application de règles de mappage sur des fragments de modèles AlbertII et du sous-ensemble définissant des relations de composition et d'assemblage

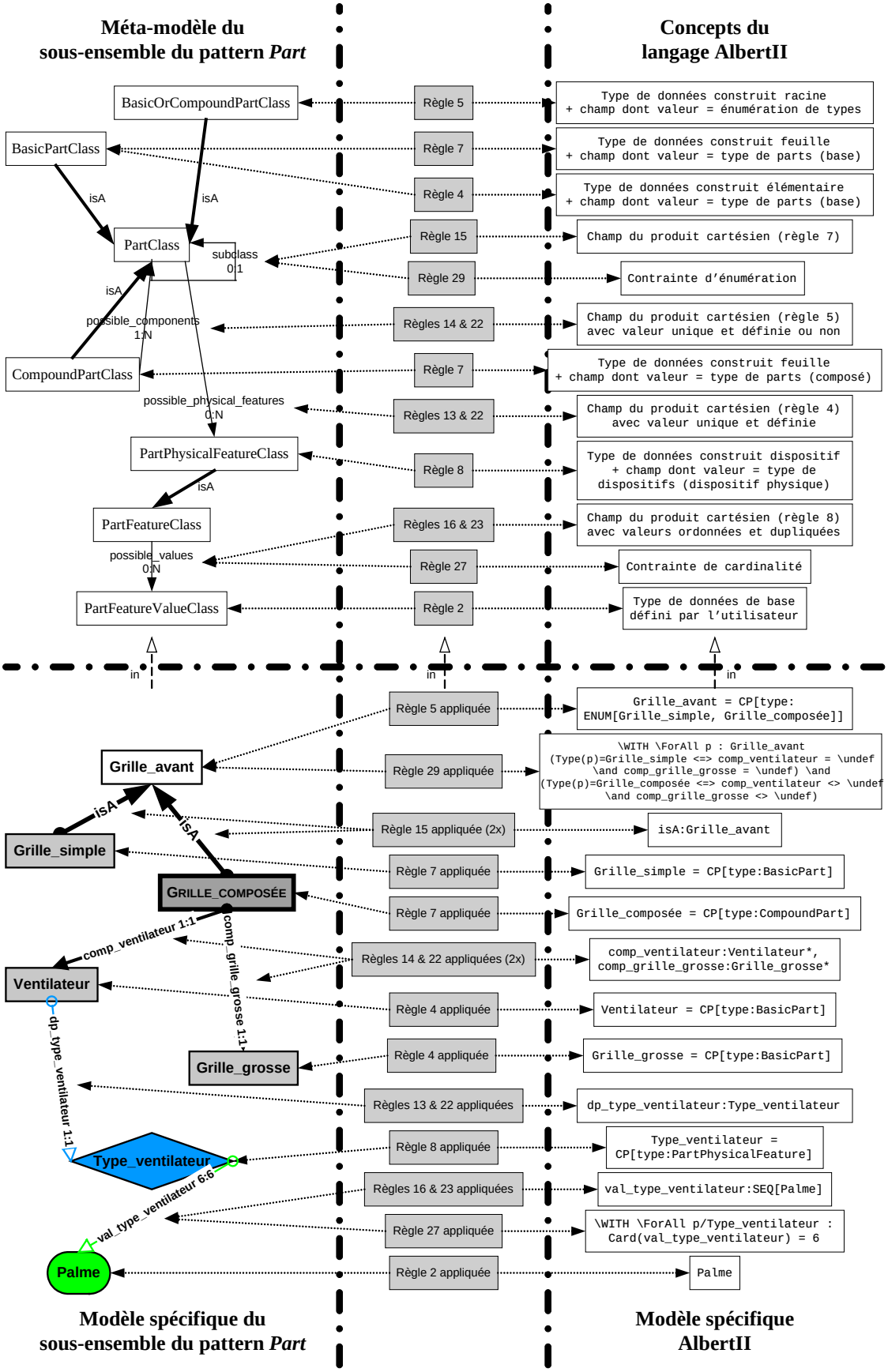


Figure E.2 : Exemple d'application de règles de mappage sur des fragments de modèles AlbertII et du sous-ensemble définissant des relations de spécialisation et de composition

Annexe F

Code du projet VBA du modèle "Systèmes manufacturiers"

F.1. Présentation du code du projet VBA

Tous les projets Visio de modèle graphique de "parts" contiennent le même projet VBA car il est hérité du modèle de dessin "Systèmes manufacturiers". Son fonctionnement est transparent pour l'utilisateur qui n'a pas l'opportunité de le modifier. Il remplit invariablement une triple fonction :

- vérifier la majorité des contraintes appartenant à la première catégorie;
- personnaliser l'espace de travail, notamment l'interface utilisateur;
- traduire les modèles graphiques de "parts" en langage AlbertII.

Au sein du logiciel Visio, le code du projet VBA du modèle "Systèmes manufacturiers" est réparti dans trois répertoires différents :

- le répertoire "Visio Objets" qui contient un module de classe spécial et unique appelé "ThisDocument";
- le répertoire "Modules de classe" qui renferme un module de classe;
- le répertoire "Modules" qui englobe sept modules standards.

Le module de classe "ThisDocument" possède deux particularités intéressantes. La première, c'est que, contrairement aux variables des méthodes et des autres modules dont la durée de vie est limitée au temps d'exécution du code de ces méthodes ou de ces modules, les variables de "ThisDocument" possèdent une durée de vie équivalente à celle du projet : elles conservent leur valeur tant que le projet est ouvert dans le programme Visio et que l'exécution du projet n'a pas connu d'arrêt, auquel cas elles sont réinitialisées. Cette spécificité nous permet d'utiliser les mêmes variables durant toute la construction du modèle graphique de "parts", depuis l'ouverture ou la création du document (projet) jusqu'à sa fermeture.

La seconde particularité est liée à la coexistence de deux types d'événements : ceux qui sont activés parce qu'ils sont associés à un gestionnaire prédéfini par les concepteurs du logiciel Visio et ceux qui ne le sont pas parce que leur gestionnaire doit être défini par l'utilisateur.[54] Dans ce cas, l'événement doit être ajouté au projet VBA (à la collection d'événements d'un objet Visio) et un module de classe doit implémenter l'interface associée à cet événement. Les événements sont attachés aux objets Visio tels que les documents, les pages, les formes ou les cellules. Pour utiliser un événement relatif à la modification d'une forme, par exemple, nous devons créer une variable objet qui représente cette forme et lui associer le gestionnaire d'événements. Si nous souhaitons utiliser cet événement pour chaque forme du dessin, nous devons représenter chacune d'entre elles par une variable objet

[54] Nous employons le terme "activés" parce que le gestionnaire se déclenche dès que l'événement se produit même s'il n'est pas utilisé dans le projet VBA. Pour éviter de consommer trop de ressources et donc de ralentir le programme, les événements susceptibles de se déclencher fréquemment ne sont pas implémentés à l'avance par les concepteurs du logiciel.

différente. Mais le même événement peut être attaché à un document et, dans ce cas, il suffit de créer une variable objet représentant le document pour que le gestionnaire se déclenche chaque fois qu'une forme du document est modifiée. Or, nous savons, d'une part, que l'objet "Document" possède des événements activés intéressants et que, d'autre part, "ThisDocument" est à la fois un module de classe et un objet Visio qui référence l'objet "Document" (représentant le projet qui est aussi le modèle graphique de "parts") et qui peut servir de variable objet. Par conséquent, nous utiliserons prioritairement les événements de l'objet "Document" parce qu'ils sont déjà activés et parce qu'ils peuvent être utilisés directement dans le module de classe "ThisDocument" sans que nous ayons besoin de créer des variables objets. Notre intérêt est en effet d'employer des événements activés pour ne pas devoir les définir (implémenter leur interface) et des événements liés au document (donc aussi à "ThisDocument" qui le référence) pour ne pas devoir créer des variables objets.

Nous attribuons à chaque classe de parts ou d'états du modèle un numéro unique qui sert d'identifiant. Ce numéro est placé dans le troisième champ de données (*Data3*) des formes qui représentent ces classes. Il est employé lors de la traduction du modèle en langage AlbertII : il sert d'indice au tableau temporaire contenant les données sur les classes. Un tableau permanent – parce que déclaré dans le module de classe "ThisDocument" – recense les informations relatives à cette numérotation : le numéro sert d'indice au tableau qui contient pour chaque classe son nom, ainsi que le nombre de formes qui la représente. Ce tableau est régulièrement mis à jour lors de la construction du modèle.

Au sein du projet VBA, nous recourons à cinq événements qui se déclenchent à des moments différents :

- "DocumentCreated" : après la création d'un document;
- "DocumentOpened" : après l'ouverture d'un document (préexistant);
- "ShapeAdded" : après l'ajout d'une forme sur un document;
- "FormulaChanged" : après le changement de la formule d'une cellule d'un document;
- "BeforeShapeDelete" : avant la suppression d'une forme.

Les trois premiers événements sont activés et sont utilisés dans le module de classe spécial "ThisDocument". L'interface qui gère les deux derniers événements est implémentée dans le module de classe "Classe1".

Voici une présentation succincte de l'algorithme du projet VBA au travers de ses neuf modules.

Le "Module6" contient une procédure qui ajoute trois boutons à la barre d'outils de fenêtre Visio : le premier permet de traduire le modèle graphique de "parts" en langage AlbertII, le deuxième d'ajouter une page au projet et le troisième de supprimer la page active du document si elle est vide.

Le "Module7" renferme les deux procédures qui implémentent les deux boutons permettant d'ajouter et de supprimer des pages.

Le module de classe "ThisDocument" contient les actions associées aux trois événements activés et liés au "Document". Le premier, "DocumentCreated", ajoute les trois boutons du "Module6" à la barre d'outils, ainsi que les événements "BeforeShapeDelete" et "FormulaChanged" à la collection d'événements du "Document". Le deuxième, "DocumentOpened", vérifie en plus la syntaxe et l'unicité des noms des classes présentes au sein du modèle graphique de "parts" si l'utilisateur le souhaite. Il attribue également un numéro à chacune des classes de parts et d'états. Le troisième, "ShapeAdded", teste l'unicité et la syntaxe du nom de la classe représentée par la forme nouvellement créée et lui attribue un numéro s'il s'agit d'une classe de parts ou d'états.

La "Classe1" implémente l'interface "IVisEventPRoc" qui permet de gérer les notifications des événements "BeforeShapeDelete" et "FormulaChanged" liés au "Document" et activés dans le module de classe "ThisDocument". Cette interface est décrite dans la bibliothèque de types Visio. Son unique fonction est implémentée par une fonction correspondante dans la "Classe1". L'action associée à l'événement " BeforeShapeDelete" est la suppression du numéro de la forme dans le tableau permanent : le compteur de la classe représentée par la forme est mis à jour. L'action associée à l'événement "FormulaChanged" consiste soit à vérifier la syntaxe et l'unicité du nom de la classe (contrainte n° 1) et à éventuellement changer son numéro si le nom de la classe est modifié, soit à s'assurer que les valeurs ordonnées d'un dispositif puissent être dupliquées si le booléen d'ordonnancement ou de duplication est modifié (contrainte n° 25).

Lorsqu'un utilisateur change le nom d'une classe, il déclenche un événement qui exécute un code vérifiant la syntaxe et l'unicité du nouveau nom. Avant d'utiliser Automation pour gérer cet événement, nous avons tenté de le faire au moyen de la feuille ShapeSheet. La section "Evénements" de cette feuille contient en effet des formules qui définissent les événements d'une forme. L'une d'entre elle exécute le contenu de la cellule "LeTexte" lorsque la composition du texte de la forme change. Or, comme nous l'avons mentionné dans la section consacrée à la correspondance entre le sous-ensemble du pattern *Part* et les concepts Visio, le nom de la classe qui se trouve dans les propriétés personnalisées de la forme est aussi inséré dans le bloc de texte. Nous avons donc placé le nom de la macro chargée d'exécuter la vérification dans la cellule "LeTexte", soit directement, soit à l'intérieur de la fonction "RUNADDON" dont le rôle consiste à transmettre un code à exécuter (la macro) vers le projet VBA du document. La référence de la cellule contenant le nouveau nom de la classe, sinon le nom lui-même, devait obligatoirement être passée en paramètre afin que le code puisse effectuer les vérifications.

Mais le projet n'est jamais parvenu à trouver la macro : dans ce cas, il n'effectuait aucune opération et ne renvoyait pas d'erreur. En effet, contrairement à ce que démontraient les exemples fournis par l'aide de Visio, le programme était incapable d'identifier une macro qui contenait des arguments. En relation avec cette incohérence, la base de connaissances Microsoft en ligne reconnaissait qu'une macro nécessitant des arguments n'apparaissait pas dans les boîtes de dialogue "Affecter une macro" et "Macro" mais pouvait néanmoins être appelée par une ligne de commande avec le nombre d'arguments correct.[55] [Base 2003] Or, nous avons vainement tenté de le faire en reproduisant les exemples accompagnant ce commentaire.

[55] Les boîtes de dialogue "Affecter une macro" et "Macro", accessibles depuis le menu "Outils", permettent d'exécuter une macro manuellement à partir de l'application Visio.

Le "Module1" contient une procédure unique qui vérifie si le modèle graphique de "parts" respecte les contraintes de première catégorie puis qui le traduit en langage AlbertII. Lorsqu'une contrainte n'est pas honorée, l'utilisateur est averti par une boîte de dialogue qui présente des données utiles à la correction et qui concernent, notamment, la contrainte et la classe fautive avec, parfois, la page où elle se situe. Voir la figure F.1 dont l'exemple est emprunté à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D. Dans ce cas, la procédure est aussitôt annulée.

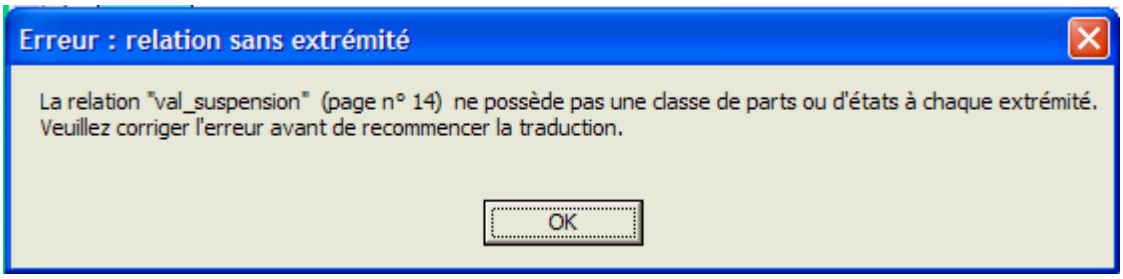


Figure F.1 : Exemple de boîte de dialogue pour la correction d'une contrainte sur la classe d'attributs dispositifs de valeur *val_suspension* dont il manque une classe de destination

L'algorithme de la procédure de vérification des contraintes et de traduction du modèle en langage AlbertII, qui fait appel aux méthodes du "Module2", du "Module3", du "Module4" et du "Module5", procède en cinq étapes :

- contrôle des contraintes sur les classes d'attributs et sur les commentaires ("Module2");
- enregistrement dans un tableau provisoire des informations sur les classes nécessaires à la traduction. Les informations utiles sont : le numéro de la classe qui sert d'indice au tableau, le nom de la classe, le nom de la méta-classe dont elle est l'instance ou de la classe spéciale dont elle est le sous-ensemble, ainsi que les commentaires et les données relatives aux classes d'attributs (pour chaque classe d'attributs : son nom et celui de la méta-classe dont elle est l'instance ou de la classe spéciale dont elle est le sous-ensemble, les bornes de la relation, les booléens de duplication et d'ordre, et le numéro de la classe destination);
- examen des contraintes particulières ("Module5");
- vérification des contraintes sur les classes de parts et d'états ("Module3" et "Module5");
- génération du code AlbertII dans l'ordre suivant ("Module4") :
  - création du fichier dont le nom et la localisation peuvent être choisis par l'utilisateur. Une routine est chargée de gérer les erreurs sur le chemin d'accès au fichier afin d'éviter le "plantage" du programme : un message informe l'utilisateur de l'échec, tandis que la procédure de traduction est annulée. Voir la figure F.2 dont l'exemple est emprunté à l'exemple de modélisation du tracteur Massey Ferguson présenté à l'annexe D;
  - écriture du titre du code dans le fichier;
  - écriture des types de données de base et de leurs commentaires dans le fichier;
  - écriture des types de données construits accompagnés de leurs contraintes et de leurs commentaires dans le fichier dans l'ordre suivant : les types des classes de parts élémentaires, les types des super classes de parts racines, les types des classes de parts sous-ensembles et super classes, les types des sous-classes de parts feuilles,

les types des classes de dispositifs physiques, les types des classes de dispositifs géométriques, les types des classes de dispositifs de fixation, les types des classes de dispositifs de contiguïté et les types des classes de dispositifs d'enclos;

- fermeture du fichier.

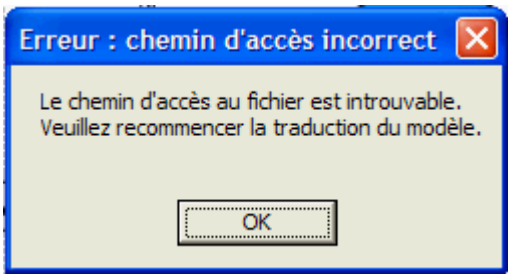


Figure F.2 : Exemple de message en cas d'erreur sur le chemin d'accès au fichier lors de la traduction du modèle graphique de "parts" en langage AlbertII

Le "Module5" contient une procédure qui vérifie les contraintes sur les classes d'attributs composants, contenus, sous-classes, dispositifs physiques, dispositifs géométriques, dispositifs de contiguïté et dispositifs d'enclos (contraintes n° 5, n° 12, n° 13 et n° 14) et une procédure qui examine si une part composée ou une pile peut posséder plusieurs composants et si un conteneur peut renfermer plusieurs contenus (contrainte n° 22). La première est appelée par le "Module1" et la seconde par le "Module3". Toutes les deux sont placées dans un module différent parce qu'elles font intervenir des variables statiques qui ne peuvent pas être déclarées dans le module appelant. Ces variables conservent leur valeur entre les appels de méthode – qui sont ici des méthodes récursives – c'est-à-dire durant toute la durée de l'exécution du code du module. Dans le cas des variables non statiques, un espace de stockage leur est attribué à chaque appel et est libéré à chaque sortie.

Le code du projet VBA du modèle "Systèmes manufacturiers" qui est proposé dans les sections suivantes s'articule autour des neuf modules selon cet ordre :

- le module de classe spécial "ThisDocument";
- le module de classe "Classe1";
- les modules standards du "Module1" au "Module7".

La présentation respecte la structure et la mise en forme du code dans l'éditeur Visio : les commentaires sont en vert, les mots-clés du langage Visual Basic sont en bleu et une ligne horizontale sépare la partie déclaration et les différentes méthodes (procédures et fonctions) de chaque module.

F.2. Module de classe spécial "ThisDocument"

```
'Module de classe nommé "ThisDocument". _
Lorsqu'il est référencé depuis le code dans le projet, l'objet "ThisDocument" renvoie _
une référence à l'objet "Document" du projet (soit au fichier de dessin ou de modèle). _
Ce code est attaché au fichier de modèle Visio (".vst") "Systèmes manufacturiers". _
Il teste l'unicité des noms de classes de parts, d'états et d'attributs _
représentées par les formes au sein du modèle graphique de "parts". _
Il attribue également un numéro à chaque classe de parts et d'états. _
Il ajoute les événements "BeforeShapeDelete" et "FormulaChanged" à la collection _
d'événements du document. _
Il ajoute aussi à la barre d'outils trois boutons permettant de traduire le modèle graphique _
de "parts" en langage AlbertII et d'ajouter/supprimer des pages au document.

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text

'Tableau à 2 dimensions comprenant les données relatives à la numérotation _
des classes de parts et d'états. _
Dimension1 : l'indice identifie un numéro compris entre 1 et MAXTAB. _
Dimension2 : l'indice identifie les informations suivantes sur la classe : _
(x,0) : nombre de formes représentant la classe dans le modèle graphique; _
(x,1) : nom de la classe; _
La cellule (0,0) indique le nombre de classes contenues dans le tableau.
Dim tabPart(0 To MAXTAB, 0 To 1) As String
'Variable indiquant le numéro le plus bas qu'il soit possible d'attribuer à une classe
Dim numBas As Integer
'Variable contenant le message adressé à l'utilisateur
Dim message As String
'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
Dim typeErreur As String
'Variable indispensable à la fonction "MsgBox" mais sans effet dans ce code (à une exception près)
Dim z As Integer
```

```
'Le gestionnaire d'événements "DocumentCreated" est exécuté à la création d'un document.
'Il ajoute les événements "BeforeShapeDelete" et "FormulaChanged" à la collection
'd'événements du document.
'Il ajoute également à la barre d'outils les trois boutons cités ci-dessus.

Private Sub Document_DocumentCreated(ByVal doc As Visio.IVDocument)

    Debug.Print "document - création"

    'Lance la macro chargée d'ajouter les boutons de traduction/page à la barre d'outils
    ajouterBoutons
```

```
    'Lance la macro chargée de créer les événements "BeforeShapeDelete" et "FormulaChanged"
    créerEvénements

    'Initialise le tableau comprenant les numéros des classes de parts et d'états
    initialiserTabPart

End Sub
```

```
'Le gestionnaire d'événements "DocumentOpened" est exécuté à l'ouverture d'un document.
'Il ajoute les événements "BeforeShapeDelete" et "FormulaChanged" à la collection
'd'événements du document.
'Il ajoute à la barre d'outils les trois boutons cités ci-dessus.
'Il teste l'unicité des noms de classes de parts, d'états et d'attributs des formes
'présentes au sein du modèle graphique de "parts" si l'utilisateur le souhaite.
'Il attribue également un numéro à chaque classe de parts et d'états.

Private Sub Document_DocumentOpened(ByVal doc As Visio.IVDocument)

    'Nombre de pages du document actif
    Dim nombrePages As Integer
    'Compteur du nombre de pages
    Dim x As Integer
    'Collection comprenant un objet shape pour chaque forme de la page active
    Dim collectionFormes As Visio.Shapes
    'Nombre de formes (parts, états et attributs) de la page active
    Dim nombreFormes As Integer
    'Compteur de formes (parts, états et attributs)
    Dim i As Integer

    Debug.Print "document - ouverture"

    'Demande à l'utilisateur si l'unicité des noms de classes doit être vérifiée
    message = "Souhaitez-vous tester l'unicité des noms des classes déjà présentes " _
        & "dans le modèle """ + ActiveDocument.Name + """ ?"
    typeErreur = "Test d'unicité des classes"
    z = MsgBox(message, vbYesNo + vbInformation + vbDefaultButton2, typeErreur)

    'Lance la macro chargée d'ajouter les boutons de traduction/page à la barre d'outils
    ajouterBoutons

    'Lance la macro chargée de créer les événements "BeforeShapeDelete" et "FormulaChanged"
    créerEvénements

    'Initialise le tableau comprenant les numéros des classes de parts et d'états
    initialiserTabPart

    'Renvoie le nombre de pages du document
    nombrePages = Visio.ActiveDocument.Pages.Count
```

```
    'Pour toutes les pages du document
    For x = 1 To nombrePages

        'Renvoie la collection de formes de la page
        Set collectionFormes = Visio.ActiveDocument.Pages(x).Shapes
        'Renvoie le nombre de formes de la collection
        nombreFormes = collectionFormes.Count

        'Teste l'unicité du nom de chaque classe si nécessaire puis leur attribue un numéro
        For i = 1 To nombreFormes
            'Elimine les classes d'attributs sous-classes qui ne possèdent pas de nom et _
            dont la forme est la seule sans section "Propriétés personnalisées" (n° 243)
            If Not (0 = collectionFormes.Item(i).SectionExists(243, 0)) Then
                'Teste si l'unicité des noms de classe doit être vérifiée
                If (z = 6) Then
                    vérifierUnicitéNom collectionFormes.Item(i), False
                End If
                attribuerNuméro collectionFormes.Item(i)
            End If
        Next i

    Next x
```

```
End Sub

'Le gestionnaire d'événements "ShapeAdded" est exécuté chaque fois qu'une forme est
'ajoutée sur la page de dessin, que cette forme soit insérée à partir d'un gabarit,
'dessinée avec un outil de dessin ou collée à partir du Presse-papiers.
'Il teste l'unicité du nom de la classe représentée par cette forme.
'Il lui attribue également un numéro si elle représente une classe de parts ou d'états.

Private Sub Document_ShapeAdded(ByVal forme As Visio.IVShape)

    Debug.Print "forme - création"

    'Teste l'unicité du nom de chaque classe puis leur attribue un numéro. _
    Elimine les classes d'attributs sous-classes qui ne possèdent pas de nom et _
    dont la forme est la seule sans section "Propriétés personnalisées" (n° 243)
    If Not (0 = forme.SectionExists(243, 0)) Then
        vérifierUnicitéNom forme, True
        attribuerNuméro forme
    End If

End Sub
```

```
'Ajoute les événements "BeforeShapeDelete" et "FormulaChanged" à la collection
'd'événements du document.  La classe réceptrice de ces événements est la "Classe1".

Private Sub créerEvénements()

    'Collection d'événements du document
    Dim collectionEvénements As Visio.EventList
    'Référence à l'interface OLE "Classe1" sur l'objet qui reçoit les notifications _
    des événements "BeforeShapeDelete" et "FormulaChanged"
    Dim refClasse1 As Classe1
    'Variable événement indispensable pour créer les deux événements précités
    Dim événement As Visio.Event

    'Récupère la collection d'événements du document
    Set collectionEvénements = Visio.ActiveDocument.EventList

    'Crée une instance de la classe "Classe1" chargée de gérer les notifications _
    des événements "BeforeShapeDelete" et "FormulaChanged"
    Set refClasse1 = New Classe1

    'Ajoute les événements respectifs précités dans le document actif
    Set événement = collectionEvénements.AddAdvise(visEvtDel + visEvtShape, _
        refClasse1, "", "Forme supprimée ...")
    Set événement = collectionEvénements.AddAdvise(visEvtMod + visEvtFormula, _
        refClasse1, "", "Formule modifiée ...")

End Sub
```

```
'Initialise le tableau qui contiendra les numéros des classes de parts et d'états

Private Sub initialiserTabPart()

    'Compteur et indice du tableau
    Dim i As Integer

    For i = 0 To MAXTAB
        tabPart(i, 0) = 0
        tabPart(i, 1) = ""
    Next i
    numBas = 1

End Sub
```

```
'Attribue un numéro à une classe de parts ou d'états
'forme : forme du projet représentant la classe de parts ou d'états

Public Sub attribuerNuméro(forme As Visio.Shape)
```

```
'Indice du tableau permanent contenant les numéros de parts et d'états
Dim i As Integer
'Compteur du nombre de classes de parts et d'états
Dim x As Integer
'Booléen indiquant l'existence d'une classe de parts ou d'états
Dim existenceClasse As Boolean

'Teste s'il s'agit d'une classe de parts ou d'états
If (forme.Layer(1).Name Like "Part*Class") Then

    'Initialise les variables indice du tableau (i) et nombre de classes (x)
    i = 1
    x = 0
    existenceClasse = False

    'Le traitement est différent si le nom de classe est un type prédéfini ou non
    Select Case forme.Cells("Prop.Nom").ResultStr(0)

    'Traite les noms de classe aux types prédéfinis
    Case "chaîne de caractères", "caractère", "booléen", "entier", "rationnel", "durée"
        'Attribue un nouveau numéro à la nouvelle classe
        If Not (choisirNuméro(forme)) Then
            'Force l'arrêt de l'exécution du code si un numéro n'a pu être attribué
            Exit Sub
        End If

    'Traite les noms de classe différents des types prédéfinis
    Case Else
        'Tant que tous les numéros de classe n'ont pas été examinés et _
        qu'une classe identique à celle de la forme en argument n'a pas été trouvée ...
        While (x < tabPart(0, 0)) And Not (existenceClasse)
            '... teste si le numéro est attribué à une classe ...
            If (tabPart(i, 0) <> 0) Then
                '... teste si la classe est identique à celle de la forme en argument. _
                Le caractère unique du nom a déjà été vérifié.
                If (tabPart(i, 1) Like forme.Cells("Prop.Nom").ResultStr(0)) Then
                    'Attribue à la forme le numéro de la classe existente
                    forme.Data3 = i
                    'Met à jour le compteur du nombre de formes représentant cette classe
                    tabPart(i, 0) = tabPart(i, 0) + 1
                    existenceClasse = True
                End If
                'Incrémente le nombre de classes
                x = x + 1
            End If
            'Incrémente l'indice du tableau de classes
            i = i + 1
        Wend
        'Teste si la forme a reçu le numéro d'une classe préexistente ...
        If Not (existenceClasse) Then
```

```
                    '... sinon attribue un nouveau numéro à la nouvelle classe
                    If Not (choisirNuméro(forme)) Then
                        'Force l'arrêt de l'exécution du code si un numéro n'a pu être attribué
                        Exit Sub
                    End If
                End If

        End Select

    End If

End Sub
```

```
'Décrémente le compteur de formes associé à une classe de parts ou d'états.
'num : numéro de la classe dont le compteur de formes doit être décrémenté.

Public Sub supprimerNuméro(num As Integer)

    'Décrémente le compteur au numéro du tableau contenant les numéros de parts et d'états
    tabPart(num, 0) = tabPart(num, 0) - 1
    'Teste si aucune forme ne porte plus ce numéro de classe ...
    If (tabPart(num, 0) = 0) Then
        '... auquel cas la numérotation et le compteur de classes doivent être mis à jour
        tabPart(num, 1) = ""
        tabPart(0, 0) = tabPart(0, 0) - 1
        'Teste si le numéro supprimé est inférieur au numéro le plus bas attribuable
        If (num < numBas) Then
            numBas = num
        End If
    End If

End Sub
```

```
'Vérifie l'unicité et la syntaxe du nom d'une la classe de parts, d'états ou d'attributs.
'forme : forme représentant la classe dont le nom est à vérifier.
'La valeur du booléen "avertissement" indique si l'utilisateur est prévenu lorsqu'une même
'classe est définie sur plusieurs pages (sa valeur est à "faux" à l'ouverture du document).

Public Sub vérifierUnicitéNom(forme As Visio.Shape, avertissement As Boolean)

    'Nombre de pages du document actif
    Dim nombrePages As Integer
    'Collection comprenant un objet shape pour chaque forme d'une page
    Dim collectionFormes As Visio.Shapes
    'Nombre de formes (parts, états et attributs) dans une page (feuille)
    Dim nombreFormes As Integer
    'Compteur du nombre de pages
    Dim x As Integer
```

```
'Compteur du nombre de formes (parts, états et attributs)
Dim i As Integer
'Variables utilisées pour modifier le nom de certaines classes
Dim nouveauNom As String, nom As String
'Longueur d'une chaîne de caractères (nom de la classe)
Dim longueur As Integer

'Renvoie la longueur du nom de la classe
longueur = Len(forme.Cells("Prop.Nom").ResultStr(0))
'Teste si le nom de la classe n'est pas nul (pour éviter un plantage lors du test suivant)
If Not (longueur = 0) Then
    'Teste si le nom de la classe commence ou se termine par un espace
    If (Mid(forme.Cells("Prop.Nom").ResultStr(0), 1, 1) Like " ") Or _
        (Mid(forme.Cells("Prop.Nom").ResultStr(0), longueur, 1) Like " ") Then
        'Envoie un message à l'utilisateur signalant qu'il a encodé un nom incorrect
        message = "Le nom d'une classe ne peut débuter ou se terminer par un espace." _
            & Chr(10) & Chr(10) & "Veuillez introduire un nouveau nom." & Chr(10) & Chr(10)
        typeErreur = "Erreur : nom de classe incorrect"
        'Initialise la variable qui contiendra le nom renvoyé par l'utilisateur
        nouveauNom = ""
        'Affecte la valeur renvoyée par la fonction InputBox à nouveauNom
        nouveauNom = InputBox(message, typeErreur)
        'Remplace l'ancien nom de classe par le nouveau. _
        Place des guillemets obligatoires aux extrémités.
        nom = """" & nouveauNom & """"
        forme.Cells("Prop.Nom").Formula = nom
        'Relance la vérification de l'unicité et de la syntaxe du nom de la classe
        vérifierUnicitéNom forme, True
    End If
End If

'Renvoie le nom de la classe
Select Case forme.Cells("Prop.Nom").ResultStr(0)

'Teste si le nom de la classe est vide, ce qui est interdit
Case ""
    'Teste si la classe n'est pas un commentaire (qui ne porte pas de nom)
    If Not (forme.Layer(1).Name Like "CommentClass") Then
        'Envoie un message à l'utilisateur signalant qu'il a encodé un nom vide
        message = "Le nom d'une classe de parts, d'états ou d'attributs ne peut être vide." _
            & Chr(10) & Chr(10) & "Veuillez introduire un nouveau nom." & Chr(10) & Chr(10)
        typeErreur = "Erreur : nom de classe vide"
        'Initialise la variable qui contiendra le nom renvoyé par l'utilisateur
        nouveauNom = ""
        'Affecte la valeur renvoyée par la fonction InputBox à nouveauNom
        nouveauNom = InputBox(message, typeErreur)
        'Remplace l'ancien nom de classe par le nouveau. _
        Place des guillemets obligatoires aux extrémités.
        nom = """" & nouveauNom & """"
        forme.Cells("Prop.Nom").Formula = nom
```

```
        'Relance la vérification de l'unicité et de la syntaxe du nom de la classe
        vérifierUnicitéNom forme, True
    End If

'Teste si le nom de la classe est équivalent au mot-clé AlbertII "UNDEF", ce qui est interdit
Case "undef"
    'Teste si la classe n'est pas un commentaire (qui ne porte pas de nom)
    If Not (forme.Layer(1).Name Like "CommentClass") Then
        'Envoie un message à l'utilisateur signalant qu'il a encodé un nom interdit
        message = "Vous ne pouvez pas utiliser le mot-clé AlbertII ""UNDEF"" comme nom pour " _
            & "une classe." & Chr(10) & Chr(10) & _
            "Veuillez introduire un nouveau nom." & Chr(10) & Chr(10)
        typeErreur = "Erreur : nom de classe incorrect"
        'Initialise la variable qui contiendra le nom renvoyé par l'utilisateur
        nouveauNom = ""
        'Affecte la valeur renvoyée par la fonction InputBox à nouveauNom
        nouveauNom = InputBox(message, typeErreur)
        'Remplace l'ancien nom de classe par le nouveau. _
        Place des guillemets obligatoires aux extrémités.
        nom = """" & nouveauNom & """"
        forme.Cells("Prop.Nom").Formula = nom
        'Relance la vérification de l'unicité et de la syntaxe du nom de la classe
        vérifierUnicitéNom forme, True
    End If

'Teste si le nom de la classe est un type prédéfini (qui peut ne pas être unique)
'Le type prédéfini ne concerne que les classes d'états
Case "chaîne de caractères", "caractère", "booléen", "entier", "rationnel", "durée"
    'Teste si le nom n'appartient pas à une classe d'états
    If Not (forme.Layer(1).Name Like "PartStateClass") Then
        'Envoie un message à l'utilisateur signalant qu'il a encodé un type prédéfini
        message = "Le nom d'une classe de parts ou d'attributs ne peut être un type prédéfini." _
            & Chr(10) & "Les types prédéfinis sont : ""chaine de caractères"", " _
            & """caractère"", ""booléen"", ""entier"", ""rationnel"" et ""durée""." _
            & Chr(10) & Chr(10) & "Veuillez introduire un nouveau nom." & Chr(10) & Chr(10)
        typeErreur = "Erreur : type prédéfini pour classe de parts ou d'attributs"
        'Initialise la variable qui contiendra le nom renvoyé par l'utilisateur
        nouveauNom = ""
        'Affecte la valeur renvoyée par la fonction InputBox à nouveauNom
        nouveauNom = InputBox(message, typeErreur)
        'Remplace l'ancien nom de classe par le nouveau. _
        Place des guillemets obligatoires aux extrémités.
        nom = """" & nouveauNom & """"
        forme.Cells("Prop.Nom").Formula = nom
        'Relance la vérification de l'unicité et de la syntaxe du nom de la classe
        vérifierUnicitéNom forme, True
    End If
```

```
'Teste si le nom de la classe est unique
Case Else
    'Renvoie le nombre de pages du document
    nombrePages = Visio.ActiveDocument.Pages.Count
    'Vérifie dans toutes les pages du document
    For x = 1 To nombrePages
        'Renvoie la collection de formes de la page
        Set collectionFormes = Visio.ActiveDocument.Pages(x).Shapes
        'Renvoie le nombre de formes de la collection
        nombreFormes = collectionFormes.Count
        'Vérifie le nom de toutes les classes de la page
        For i = 1 To nombreFormes
            'Teste si la classe possède un nom (si pas classe d'attributs sous-classe)
            If Not (collectionFormes.Item(i).Layer(1) Like "SubClass") Then
                'Teste si le nom de la classe est identique
                If (forme.Cells("Prop.Nom").ResultStr(0) Like _
                    collectionFormes.Item(i).Cells("Prop.Nom").ResultStr(0)) Then
                    'Teste si la classe se trouve sur la même page et _
                    si elle est représentée par une forme différente. _
                    Une page ne peut contenir plusieurs fois la même classe.
                    If (x Like forme.ContainingPage) And _
                        Not (forme.id Like collectionFormes.Item(i).id) Then
                        'Envoie un message à l'utilisateur
                        message = "La page active n° " & x & _
                            " possède déjà une classe portant le nom " _
                            & forme.Cells("Prop.Nom").Formula & "." _
                            & Chr(10) & "Or une page ne peut contenir plusieurs fois " & _
                            "le même nom de classe." & Chr(10) & Chr(10) & _
                            "Veuillez introduire un nouveau nom." & Chr(10) & Chr(10)
                        typeErreur = "Erreur : classe plusieurs fois sur la même page"
                        'Initialise la variable qui contiendra le nom renvoyé par l'utilisateur
                        nouveauNom = ""
                        'Affecte la valeur renvoyée par la fonction InputBox à nouveauNom
                        nouveauNom = InputBox(message, typeErreur)
                        'Remplace l'ancien nom de classe par le nouveau. _
                        Place des guillemets obligatoires aux extrémités.
                        nom = """" & nouveauNom & """"
                        forme.Cells("Prop.Nom").Formula = nom
                        'Relance la vérification de l'unicité et de la syntaxe du nom
                        vérifierUnicitéNom forme, True
                    Else
                        'Teste si la classe se trouve sur une page différente
                        If Not (x Like forme.ContainingPage) Then
                            'Teste si la méta-classe ou la classe spéciale est différente
                            If Not (forme.Data1 Like collectionFormes.Item(i).Data1) Then
                                'Teste si classe pas sous-ensemble de classe spéciale
                                If (collectionFormes.Item(i).Data1 Like "*Class") Or _
                                    (collectionFormes.Item(i).Data1 Like "possible*") Then
                                    'Envoie un message à l'utilisateur
                                    message = "Le nom " & forme.Cells("Prop.Nom").Formula & _
```

```
                    " est déjà porté par une classe qui est une instance de " & _
                    "la méta-classe """ & collectionFormes.Item(i).Data1 & _
                    """ à la page n° " & x & "." & Chr(10) & "Or " & _
                    "deux classes, instances de méta-classes différentes" _
                    & " ou sous-ensembles de classes spéciales " _
                    & "différentes, ne peuvent porter le même nom." _
                    & Chr(10) & Chr(10) & _
                    "Veuillez introduire un nouveau nom." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : nom identique pour plusieurs classes"
                'Initialise la variable qui contiendra le nom de l'utilisateur
                nouveauNom = ""
                'Affecte la valeur renvoyée par la fonction à nouveauNom
                nouveauNom = InputBox(message, typeErreur)
                'Remplace l'ancien nom de classe par le nouveau. _
                Place des guillemets obligatoires aux extrémités.
                nom = """" & nouveauNom & """"
                forme.Cells("Prop.Nom").Formula = nom
                'Relance la vérification de l'unicité et de la syntaxe du nom
                vérifierUnicitéNom forme, True
            Else
            'Classe sous-ensemble de classe spéciale
                'Envoie un message à l'utilisateur
                message = "Le nom " & forme.Cells("Prop.Nom").Formula & _
                    " est déjà porté par une classe qui est un " _
                    & "sous-ensemble de la classe spéciale """ _
                    & collectionFormes.Item(i).Data1 & _
                    """ à la page n° " & x & "." & Chr(10) & "Or " & _
                    "deux classes, instances de méta-classes différentes" _
                    & " ou sous-ensembles de classes spéciales " _
                    & "différentes, ne peuvent porter le même nom." _
                    & Chr(10) & Chr(10) & _
                    "Veuillez introduire un nouveau nom." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : nom identique pour plusieurs classes"
                'Initialise la variable qui contiendra le nom de l'utilisateur
                nouveauNom = ""
                'Affecte la valeur renvoyée par la fonction à nouveauNom
                nouveauNom = InputBox(message, typeErreur)
                'Remplace l'ancien nom de classe par le nouveau. _
                Place des guillemets obligatoires aux extrémités.
                nom = """" & nouveauNom & """"
                forme.Cells("Prop.Nom").Formula = nom
                'Relance la vérification de l'unicité et de la syntaxe du nom
                vérifierUnicitéNom forme, True
            End If
        Else
        'Page est différente et méta-classe ou classe spéciale identique
            'Teste si l'utilisateur doit être averti si classe identique
            If avertissement Then
```

```
                                        'Avertit l'utilisateur que la classe est déjà présente _
                                        dans le modèle graphique de "parts"
                                        message = "La classe " & _
                                            forme.Cells("Prop.Nom").Formula & " est déjà " _
                                            & "représentée par une forme dans le modèle " _
                                            & "à la page n° " & x & "."
                                        typeErreur = "Avertissement"
                                        z = MsgBox(message, 0, typeErreur)
                                    End If
                                    'Teste s'il s'agit d'une classe de piles
                                    If (forme.Data1 Like "PileOfParts") Then
                                        'Teste si le type de piles est identique
                                        If Not (forme.Cells("Prop.Type").ResultStr(0) Like _
                                        collectionFormes.Item(i).Cells("Prop.Type").ResultStr(0)) Then
                                            'Envoie un message à l'utilisateur
                                            message = "Une forme représente déjà la classe de " _
                                                & "piles " & forme.Cells("Prop.Nom").Formula _
                                                & " qui est de type " & _
                                            collectionFormes.Item(i).Cells("Prop.Type").ResultStr(0) _
                                                & """ à la page n° " & x & "."
                                            typeErreur = "Avertissement"
                                            z = MsgBox(message, 0, typeErreur)
                                            'Corrige automatiquement le type de piles
                                            forme.Cells("Prop.Type").Formula = _
                                            collectionFormes.Item(i).Cells("Prop.Type").Formula
                                        End If
                                    End If
                                End If
                            End If
                        End If
                    End If
                End If
            Next i
        End If
    Next x
    'Met la première lettre du nom d'une classe de parts ou d'états en majuscule.
    'Teste s'il s'agit d'une classe de parts ou d'états
    If (forme.Layer(1).Name Like "Part*Class") Then
        'Renvoie la longueur du nom de la classe
        longueur = Len(forme.Cells("Prop.Nom").ResultStr(0))
        'Renvoie la première lettre du nom de la classe
        nom = Left(forme.Cells("Prop.Nom").ResultStr(0), 1)
        'Renvoie le reste du nom de la classe
        nouveauNom = Right(forme.Cells("Prop.Nom").ResultStr(0), longueur - 1)
        'Met la première lettre en majuscule
        nom = UCase(nom)
        'Remplace l'ancien nom de classe par le nouveau. _
        Place des guillemets obligatoires aux extrémités.
        nom = """" & nom & nouveauNom & """"
        forme.Cells("Prop.Nom").Formula = nom
    End If
```

```
        'Met la première lettre du nom d'une classe d'attributs en minuscule.
        'Teste s'il s'agit d'une classe d'attributs
        If (forme.Layer(1).Name Like "AttributeClass") Then
            'Renvoie la longueur du nom de la classe
            longueur = Len(forme.Cells("Prop.Nom").ResultStr(0))
            'Renvoie la première lettre du nom de la classe
            nom = Left(forme.Cells("Prop.Nom").ResultStr(0), 1)
            'Renvoie le reste du nom de la classe
            nouveauNom = Right(forme.Cells("Prop.Nom").ResultStr(0), longueur - 1)
            'Met la première lettre en minuscule
            nom = LCase(nom)
            'Remplace l'ancien nom de classe par le nouveau. _
            Place des guillemets obligatoires aux extrémités.
            nom = """" & nom & nouveauNom & """"
            forme.Cells("Prop.Nom").Formula = nom
        End If

    End Select

End Sub
```

```
'Donne un numéro non attribué à une classe de parts ou d'états, sinon renvoie la valeur faux
'forme : forme représentant la nouvelle classe

Private Function choisirNuméro(forme As Visio.Shape)

    'Booléen indiquant si un numéro a été attribué à la nouvelle classe
    Dim existenceNuméro As Boolean
    'Indice du tableau contenant les numéros de classes de parts et d'états
    Dim i As Integer

    'Initilise les variables
    existenceNuméro = False
    i = numBas
    While Not (existenceNuméro)
        'Teste si le numéro n'a pas encore été attribué
        If (tabPart(i, 0) = 0) Then
            forme.Data3 = i
            'Complète le tableau et met à jour le compteur de classes
            tabPart(i, 0) = 1
            tabPart(i, 1) = forme.Cells("Prop.Nom").ResultStr(0)
            tabPart(0, 0) = tabPart(0, 0) + 1
            existenceNuméro = True
            numBas = i + 1
        End If
        i = i + 1
        'Teste si le tableau de classes est saturé
        If i > MAXTAB Then
            'Affiche un message d'erreur à l'utilisateur
```

```
            message = "Le nombre de formes est supérieur à la limite autorisée : " _
                & MAXTAB & "." & Chr(10) & Chr(10) & _
                "ATTENTION : la forme " & forme.Cells("Prop.Nom").Formula & _
                " ne porte pas de numéro et va être supprimée." & Chr(10) & _
                "Veuillez avertir le concepteur du logiciel d'accroître la capacité du tableau " _
                    & "permanent contenant les données relatives à la numérotation des classes " _
                    & "de parts et d'états." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : nombre de classes trop important"
            z = MsgBox(message, 0, typeErreur)
            'Supprime la forme sans numéro
            forme.Delete
            choisirNuméro = False
        End If
    Wend
    choisirNuméro = True

End Function
```

F.3. Module de classe "Classe1"

```
'Classe réceptrice nommée "Classe1". _
Implémente l'interface "IVisEventPRoc" qui permet de gérer _
les notifications des événements "BeforeShapeDelete" et "FormulaChanged".

'L'interface IVisEventPRoc contient une seule fonction avec les arguments suivants
Implements Visio.IVisEventProc

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text
```

```
'Implémente l'unique fonction de l'interface "IVisEventPRoc"

Private Function IVisEventProc_VisEventProc( _
    ByVal nEventCode As Integer, _
    ByVal pSourceObj As Object, _
    ByVal nEventID As Long, _
    ByVal nEventSeqNum As Long, _
    ByVal pSubjectObj As Object, _
    ByVal vMoreInfo As Variant) As Variant

    'Renvoie le code de l'événement qui s'est déclenché
    Select Case nEventCode

        'Le gestionnaire d'événements "BeforeShapeDelete" est exécuté chaque fois _
        qu'une forme est supprimée. _
        Il efface le numéro de la classe de parts ou d'états associée à cette forme.
        Case (visEvtDel + visEvtShape)
            'Teste si la forme représente une classe de parts ou d'états (elle possède un numéro)
            If Not (pSubjectObj.Data3 Like "") Then
                ThisDocument.supprimerNuméro pSubjectObj.Data3
            End If

        'Le gestionnaire d'événements "FormulaChanged" est exécuté chaque fois _
        qu'un changement se produit dans la formule d'une cellule du document. _
        1) Si la cellule contient le nom d'une classe : _
          Il teste l'unicité du nouveau nom de la classe de parts, d'états ou d'attributs. _
          Il efface le numéro de la classe de parts ou d'états associé à cette forme, _
          puis lui attribue un nouveau numéro. _
        2) Si la cellule contient le booléen d'ordonnancement d'une classe de dispositifs de valeur : _
          Il vérifie que les valeurs ordonnées peuvent être dupliquées. _
        3) Si la cellule contient le booléen de duplication d'une classe de dispositifs de valeur : _
          Il vérifie que les valeurs ordonnées peuvent être dupliquées.
        Case (visEvtMod + visEvtFormula)
            'Teste si la cellule est celle qui contient le nom d'une classe
            If (pSubjectObj.Name Like "Prop.Nom") Then
```

```
                'Vérifie l'unicité du nom
                ThisDocument.vérifierUnicitéNom pSubjectObj.Shape, True
                'Teste si la forme représente une classe de parts ou d'états
                If Not (pSubjectObj.Shape.Data3 Like "") Then
                    'Efface le numéro de la classe
                    ThisDocument.supprimerNuméro pSubjectObj.Shape.Data3
                    'Attribue un nouveau numéro à la classe
                    ThisDocument.attribuerNuméro pSubjectObj.Shape
                End If
            End If
            'Teste si la cellule est celle qui contient le booléen d'ordonnancement
            If (pSubjectObj.Name Like "Prop.Ordre") Then
                'Si les valeurs ou les dispositifs doivent être ordonnés ...
                If (pSubjectObj.ResultStr(0) Like "VRAI") Then
                    '... alors ils doivent aussi pouvoir être dupliqués
                    pSubjectObj.Shape.Cells("Prop.Duplication").Formula = "VRAI"
                End If
            End If
            'Teste si la cellule est celle qui contient le booléen de duplication
            If (pSubjectObj.Name Like "Prop.Duplication") Then
                'Si les valeurs ou les dispositifs doivent être ordonnés ...
                If (pSubjectObj.Shape.Cells("Prop.Ordre").ResultStr(0) Like "VRAI") Then
                    '... alors ils doivent aussi pouvoir être dupliqués
                    pSubjectObj.Shape.Cells("Prop.Duplication").Formula = "VRAI"
                End If
            End If
        Case Else
            Debug.Print "La Classe1 ne peut gérer l'événement suivant : " & nEventCode & "."
    End Select

End Function
```

F.4. Module standard "Module1"

'Module nommé "Module1". _
Traduit un modèle graphique de "parts" en un modèle AlbertII correspondant _
qui peut être stocké dans une base MiniTelos (AlbertTelosBase) via une interface Java. _
Résolution en 2 temps : _
1) Vérifier les contraintes relatives au pattern Part et construire un tableau reprenant _
les données utiles. _
--> Ce module exécute aussi les méthodes des "Module2", "Module3" et "Module5". _
2) Traduire le modèle graphique de "parts" et placer le code généré dans un fichier. _
--> Ce module exécute pour cela la méthode du "Module4".

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text

'Taille du tableau temporaire ("tabParts") contenant les classes de parts et d'états, _
du tableau temporaire ("tabComposants") contenant données sur classes de parts composantes sens large, _
du tableau temporaire ("tabDispositifs") contenant les données sur les classes de dispositifs, _
du tableau temporaire ("tabPile") contenant les données sur une classe de piles, _
du tableau temporaire ("tabEnclos") sur les classes d'attributs dispositifs de contiguïté, _
du tableau temporaire ("tabFeuilles") contenant les données sur les sous-classes feuilles, _
du tableau temporaire ("tabNoeuds") contenant les données sur les super/sous-classes noeuds et _
du tableau permanent ("tabPart") contenant les numéros de parts et d'états de "ThisDocument".
Public Const MAXTAB **As Integer** = 150

'Tableau à 3 dimensions qui contient les informations sur les classes de parts _
et d'états, ainsi que sur les commentaires et les classes d'attributs qui les concernent. _
Dimension1 : l'indice identifie une classe de parts ou d'états. _
Dimension2 : l'indice identifie la classe de parts ou d'états elle-même, _
ainsi que chaque relation/commentaire qui lui est associé. _
Dimension3 : l'indice contient les informations sur la classe ou sur l'une de ses relations _
ou sur l'un de ses commentaires. _
Particularités des cellules d'indice "0" : elles servent souvent de compteurs. _
(0,0,0) : nombre de classes de parts et d'états contenues dans le tableau (soit n ce nombre). _
(0,0,1..n) : chaque cellule contient un indice de dimension1 qui identifie une classe. _
(x,0,0) : nombre de classes d'attributs associées à la classe de parts ou d'états n° x (soit z ce nombre). _
(x,0,1-3) : contient les données relatives à la classe de parts ou d'états n° x. _
(x,0,1) : contient le nom de la méta-classe dont la classe de parts ou d'états n° x est l'instance _
ou le nom de la classe spéciale dont la classe de parts ou d'états n° x est le sous-ensemble. _
(x,0,2) : contient le nom de la classe de parts ou d'états n° x. _
(x,0,3) : contient le type de conteneurs de la classe de conteneurs n° x. _
(x,1-z,0-6) : contient les données relatives à chacune des z classes d'attributs associées à la classe n° x. _
(x,1-z,0) : contient le nom de la méta-classe dont la classe d'attributs est l'instance _
ou le nom de la classe spéciale dont la classe d'attributs est le sous-ensemble. _
(x,1-z,1) : contient le nom de la classe d'attributs (ou le commentaire). _
(x,1-z,2) : contient la borne inférieure de la cardinalité de la classe d'attributs. _
(x,1-z,3) : contient la borne supérieure de la cardinalité de la classe d'attributs. _
(x,1-z,4) : contient le n° de la classe de parts ou d'états reliée à la classe n° x. _

(x,1-z,5) : contient le booléen indiquant si les valeurs du dispositif sont dupliquées. _
(x,1-z,6) : contient le booléen indiquant si les valeurs du dispositif sont ordonnées.
Public tabParts(0 **To** MAXTAB, 0 **To** 20, 0 **To** MAXTAB) **As String**
'Tableau à 2 dimensions qui contient les infos sur les classes de parts composantes au sens large. _
Dimension1 : l'indice identifie une classe de parts composantes au sens large. _
Dimension2 : l'indice identifie le numéro d'une classe de piles, de parts composées ou de conteneurs. _
(x,0) : nombre de classes de parts composées de la classe de parts composantes n° x (soit z ce nombre). _
(x,1-z) : contient le numéro de chacune des z classes de parts composées de la classe n° x.
Public tabComposants(0 **To** MAXTAB, 0 **To** MAXTAB) **As Integer**
'Tableau qui contient les informations sur les classes de dispositifs géométriques, _
de contiguïté et d'enclos. _
Dimension1 : l'indice identifie une classe de dispositifs. _
Dimension2 : l'indice identifie les données sur la classe d'attributs géométries, _
contiguïtés ou enclos. _
(x,0) : le numéro de la classe de parts composées au sens large définie par la classe de dispositifs. _
(x,1-2) : bornes inférieure et supérieure de la classe d'attributs.
Public tabDispositifs(0 **To** MAXTAB, 0 **To** 2) **As String**
'Tableau qui contient les informations sur les classes de parts composantes qui définissent _
les classes de dispositifs de contiguïté d'une classe de piles. _
La première cellule contient le compteur de classes de parts composantes. _
Les cellules contiennent les numéro des classes de parts composantes de la classe de piles.
Public tabPile(0 **To** MAXTAB) **As Integer**
'Tableau à 2 dimensions qui contient les informations sur les classes _
d'attributs dispositifs d'enclos liées à une classe de piles ou de conteneurs. _
Dimension1 : l'indice identifie une des classes d'attributs dispositifs d'enclos. _
Dimension2 : l'indice identifie les données d'une classe d'attributs dispositifs d'enclos. _
(0,0) : nombre de classes d'attributs dispositifs d'enclos. _
(0,1) : contient le type de piles ou de conteneurs. _
(x,0-4) : contient les données d'une classe d'attributs dispositifs d'enclos. _
(x,0) : contient le numéro de la classe de parts composantes qui est la destination de la classe. _
(x,1) : borne inférieure de la classe d'attributs dispositifs d'enclos. _
(x,2) : borne supérieure de la classe d'attributs dispositifs d'enclos.
Private tabEnclos(0 **To** MAXTAB, 0 **To** 2) **As String**
'Bornes inférieures et supérieures des classes d'attributs contiguïtés d'une classe de piles
Private borneContInf **As String**, borneContSup **As String**
'Bornes inférieures et supérieures des classes d'attributs composantes d'une classe de piles
Private borneCompInf **As String**, borneCompSup **As String**
'Booléen indiquant l'échec ou la réussite de la vérification d'une contrainte sur une classe
Public vérification **As Boolean**

'Traduit un modèle graphique de "parts" (depuis un bouton sur la barre d'outils défini au "Module6")

Sub traduireEnAlbertII()

'Nombre de pages du document actif
Dim nombrePages **As Integer**
'Collection comprenant un objet shape pour chaque forme d'une page
Dim collectionFormes **As** Visio.Shapes
'Nombre de formes (parts, états et relations) dans une page

```
Dim nombreFormes As Integer
'Compteur du nombre de pages et du nombre de classes de parts et d'états
Dim x As Integer
'Compteur du nombre de formes et du nombre de classes d'attributs
Dim i As Integer
'Indice du tableau (dimension1) correspondant au numéro de la classe de parts ou d'états
Dim num As Integer
'Nombre de relations associées à une classe de parts ou d'états
Dim nombreRelations As Integer
'Compteurs utilisés à toutes les sauces
Dim y, w As Integer
'Variable contenant le message adressé à l'utilisateur
Dim message As String
'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
Dim typeErreur As String
'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
Dim z As Integer
'Longueur d'une chaîne de caractères (commentaire)
Dim longueur As Integer
'Variable contenant le commentaire d'une classe de parts ou d'états
Dim commentaire As String
'Variable contenant nom d'une classe de parts ou d'états comme type prédéfini
Dim typ As String
'Numéro d'une classe de dispositifs d'enclos
Dim numEnclos As Integer


'PREMIERE PARTIE : vérification des contraintes sur chaque classe et _
                   sauvegarde des données utiles dans un tableau ("tabParts") _
'Résolution en 21 temps : _
1) Vérifier les contraintes sur les classes d'attributs (relations) et les commentaires. _
2) Enregistrer les classes de parts, d'états et d'attributs dans le tableau "tabParts". _
3) Vérifier si le modèle graphique de "parts" n'est pas vide. _
4) Vérifier si toutes les classes d'attributs différentes portent un nom différent. _
5) Eliminer les classes d'attributs redondantes dans le tableau. _
6) Vérifier si plus d'une relation associe les deux mêmes classes. _
7) Vérifier si une classe de parts composées ou de piles est composée d'elle-même _
  ou si elle est associée plusieurs fois à une même classe. _
8) Vérifier si une classe de tampons se contient elle-même _
  ou si elle est associée plusieurs fois à une même classe. _
9) Vérifier si une classe de parts est associée plus d'une fois à la même classe _
     de dispositifs physiques. _
10) Vérifier si une classe de parts composées est associée plus d'une fois à la même _
     classe de dispositifs géométriques. _
11) Vérifier si une classe de dispositifs de contiguïté est associée plus d'une fois _
     à la même classe de parts composantes. _
12) Vérifier si une classe de dispositifs d'enclos est associée plus d'une fois _
     à la même classe de parts composantes.
'13) Vérifier la hiérarchisation des relations sous-ensembles entre les classes de parts. _
14) Met à jour le tableau des parts composantes contenant leurs parts composées respectives. _
```

```
15) Vérifier les contraintes sur les classes de parts et d'états. _
16) Vérifier que toutes les classes de parts composantes d'une classe de piles sont _
      associées à une classe de dispositifs de contiguïté de cette classe de piles. _
17) Vérifier que toutes les classes de parts composantes d'une classe de piles sont _
      associées à la classe de dispositifs d'enclos de cette classe de piles (si nécessaire).

'Renvoie le nombre de pages du document
nombrePages = Visio.ActiveDocument.Pages.Count

'Initialise les cellules du tableau de classes "tabParts" qui contiendront les compteurs _
et les données sur les classes de parts, d'états et d'attributs, _
ainsi que les cellules du tableau des classes de parts composantes "tabComposants" _
qui contiendront les compteurs des classes de piles, de parts composées ou de conteneurs, _
ainsi que les cellules du tableau "tabDispositifs" qui contiendront les numéros et les bornes _
des classes de parts composées au sens large définies par les classes de dispositifs _
géométriques, de contiguïté et d'enclos.
For i = 0 To MAXTAB
      tabParts(i, 0, 0) = 0
      tabParts(i, 0, 1) = ""
      tabParts(i, 0, 2) = ""
      tabParts(i, 0, 3) = ""
      For y = 1 To 20
            tabParts(i, y, 0) = ""
            tabParts(i, y, 1) = ""
            tabParts(i, y, 2) = ""
            tabParts(i, y, 3) = ""
            tabParts(i, y, 4) = ""
            tabParts(i, y, 5) = ""
            tabParts(i, y, 6) = ""
      Next y
      tabComposants(i, 0) = 0
      tabDispositifs(i, 0) = 0
      tabDispositifs(i, 1) = 0
      tabDispositifs(i, 2) = 0
Next i

'Initialise le "Module2" de vérification des contraintes liées au modèle de "parts"
initialiserVérifierContraintes (MAXTAB)

'Vérifie et enregistre les classes (formes) dans toutes les pages du document
For x = 1 To nombrePages

      'Renvoie la collection de formes de la page active
      Set collectionFormes = Visio.Application.ActiveDocument.Pages(x).Shapes
      'Renvoie le nombre de formes de la collection
      nombreFormes = collectionFormes.Count

      'Analyse chaque forme (commentaire et classe de parts, d'états et d'attributs) de la page
      For i = 1 To nombreFormes
```

```
'1) Lance la vérification de la classe d'attributs ou du commentaire ("Module2")
If (collectionFormes.Item(i).Layer(1).Name Like "AttributeClass") Or _
    (collectionFormes.Item(i).Layer(1).Name Like "SubClass") Then
    vérifierContraintesAttributs collectionFormes.Item(i)
Else
    If (collectionFormes.Item(i).Layer(1).Name Like "CommentClass") Then
        vérifierContraintesCommentaires collectionFormes.Item(i)
    End If
End If
'Teste la réussite ou l'échec de la vérification
If Not (vérification) Then
    'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
    Exit Sub
End If

'2) Sauvegarde les données utiles de la classe dans le tableau "tabParts"
'Renvoie le nom de la méta-classe dont la classe est l'instance
Select Case collectionFormes.Item(i).Layer(1).Name

'Instances de la méta-classe de parts, de dispositifs ou d'états
Case "PartClass", "PartFeatureClass", "PartStateClass"
    'Renvoie le numéro de la classe de parts ou d'états
    num = collectionFormes.Item(i).Data3
    'Teste si la classe n'a pas encore été enregistrée
    If (tabParts(num, 0, 1) Like "") Then
        'Enregistre la classe de parts ou d'états dans le tableau
        typ = collectionFormes.Item(i).Cells("Prop.Nom").ResultStr(0)
        tabParts(num, 0, 1) = collectionFormes.Item(i).Data1
        tabParts(num, 0, 2) = Switch( _
            typ Like "chaîne de caractères", "STRING", _
            typ Like "caractère", "CHAR", _
            typ Like "booléen", "BOOLEAN", _
            typ Like "entier", "INTEGER", _
            typ Like "rationnel", "RATIONAL", _
            typ Like "durée", "DURATION", _
            True, typ)
        'Teste s'il s'agit d'une classe de piles
        If (collectionFormes.Item(i).Data1 Like "PileOfParts") Then
            'Enregistre le type de piles
            tabParts(num, 0, 3) = collectionFormes.Item(i).Cells("Prop.Type").ResultStr(0)
        End If
        'Met à jour la liste et le compteur de classes
        tabParts(0, 0, 0) = tabParts(0, 0, 0) + 1
        tabParts(0, 0, tabParts(0, 0, 0)) = num
    End If

'Instances de la méta-classe d'attributs
Case "AttributeClass"
    'Renvoie le numéro de la classe de parts ou d'états d'origine
    num = collectionFormes.Item(i).Connects.Item(1).ToSheet.Data3
```

```
'Met à jour le compteur de relations de la classe
tabParts(num, 0, 0) = tabParts(num, 0, 0) + 1
'Renvoie le nombre de relations de la classe
nombreRelations = tabParts(num, 0, 0)
'Enregistre la relation dans le tableau
tabParts(num, nombreRelations, 0) = collectionFormes.Item(i).Data1
tabParts(num, nombreRelations, 1) = _
    collectionFormes.Item(i).Cells("Prop.Nom").ResultStr(0)
tabParts(num, nombreRelations, 2) = _
    collectionFormes.Item(i).Cells("Prop.BorneInférieure").ResultStr(0)
tabParts(num, nombreRelations, 3) = _
    collectionFormes.Item(i).Cells("Prop.BorneSupérieure").ResultStr(0)
tabParts(num, nombreRelations, 4) = _
    collectionFormes.Item(i).Connects.Item(2).ToSheet.Data3
'Teste si la relation possède les propriétés de duplication et d'ordre
If (collectionFormes.Item(i).Data1 Like "possible_values") Then
    tabParts(num, nombreRelations, 5) = _
        collectionFormes.Item(i).Cells("Prop.Duplication").ResultStr(0)
    tabParts(num, nombreRelations, 6) = _
        collectionFormes.Item(i).Cells("Prop.Ordre").ResultStr(0)
Else
    tabParts(num, nombreRelations, 5) = ""
    tabParts(num, nombreRelations, 6) = ""
End If
'Sauvegarde les numéros des classes de piles, de parts composées et de conteneurs _
dans le tableau "tabComposants"
'Teste s'il s'agit d'une classe d'attributs composants ou contenants
If (collectionFormes.Item(i).Data1 Like "possible_components") Or _
    (collectionFormes.Item(i).Data1 Like "contain") Then
    'Renvoie le numéro de la classe de parts composantes/contenues (destination)
    y = collectionFormes.Item(i).Connects.Item(2).ToSheet.Data3
    'Renvoie le numéro de la classe de parts composées au sens large (origine)
    w = collectionFormes.Item(i).Connects.Item(1).ToSheet.Data3
    'Met à jour le compteur de classes de parts composées au sens large
    tabComposants(y, 0) = tabComposants(y, 0) + 1
    'Enregistre le numéro de la classe de parts composées au sens large
    tabComposants(y, tabComposants(y, 0)) = w
End If
'Sauvegarde les données des classes d'attributs géométries, contiguïtés _
et enclos dans le tableau "tabDispositifs".
'Teste s'il s'agit d'une classe d'attributs géométries, contiguïtés ou enclos.
If (collectionFormes.Item(i).Data1 Like "possible_geometry") Or _
    (collectionFormes.Item(i).Data1 Like "contiguity") Or _
    (collectionFormes.Item(i).Data1 Like "enclosure") Then
    'Renvoie le numéro de la classe de dispositifs géométriques (destination)
    y = collectionFormes.Item(i).Connects.Item(2).ToSheet.Data3
    'Renvoie le numéro de la classe de parts composées (origine)
    w = collectionFormes.Item(i).Connects.Item(1).ToSheet.Data3
    'Enregistre le numéro de la classe de parts composées
    tabDispositifs(y, 0) = w
```

```
            'Enregistre la borne inférieure de la classe d'attributs géométries
            tabDispositifs(y, 1) = collectionFormes.Item(i).Cells("Prop.BorneInférieure").ResultStr(0)
            'Enregistre la borne supérieure de la classe d'attributs géométries
            tabDispositifs(y, 2) = collectionFormes.Item(i).Cells("Prop.BorneSupérieure").ResultStr(0)
        End If

    'Instance de la méta-classe de sous-classes
    Case "SubClass"
        'Renvoie le numéro de la classe de parts d'origine
        num = collectionFormes.Item(i).Connects.Item(1).ToSheet.Data3
        'Met à jour le compteur de relations de la classe
        tabParts(num, 0, 0) = tabParts(num, 0, 0) + 1
        'Renvoie le nombre de relations de la classe
        nombreRelations = tabParts(num, 0, 0)
        'ATTENTION : la relation sous-ensemble est toujours enregistrée _
                    à l'indice 1 de la dimension 2
        'TEMPS 1 : fait reculer d'un rang les relations de la classe d'origine
        For y = (nombreRelations - 1) To 1 Step -1
            tabParts(num, y + 1, 0) = tabParts(num, y, 0)
            tabParts(num, y + 1, 1) = tabParts(num, y, 1)
            tabParts(num, y + 1, 2) = tabParts(num, y, 2)
            tabParts(num, y + 1, 3) = tabParts(num, y, 3)
            tabParts(num, y + 1, 4) = tabParts(num, y, 4)
            tabParts(num, y + 1, 5) = tabParts(num, y, 5)
            tabParts(num, y + 1, 6) = tabParts(num, y, 6)
        Next y
        'TEMPS 2 : enregistre la relation dans la classe d'origine
        tabParts(num, 1, 0) = "isAOrigine"
        tabParts(num, 1, 1) = ""
        tabParts(num, 1, 2) = ""
        tabParts(num, 1, 3) = ""
        tabParts(num, 1, 4) = _
            collectionFormes.Item(i).Connects.Item(2).ToSheet.Data3
        tabParts(num, 1, 5) = ""
        tabParts(num, 1, 6) = ""
        'TEMPS 3 : enregistre la relation dans la classe de destination
        'Renvoie le numéro de la classe de parts de destination
        num = collectionFormes.Item(i).Connects.Item(2).ToSheet.Data3
        'Met à jour le compteur de relations de la classe
        tabParts(num, 0, 0) = tabParts(num, 0, 0) + 1
        'Renvoie le nombre de relations de la classe
        nombreRelations = tabParts(num, 0, 0)
        'Enregistre la relation dans la classe de destination
        tabParts(num, nombreRelations, 0) = "isADestination"
        tabParts(num, nombreRelations, 1) = ""
        tabParts(num, nombreRelations, 2) = ""
        tabParts(num, nombreRelations, 3) = ""
        tabParts(num, nombreRelations, 4) = _
            collectionFormes.Item(i).Connects.Item(1).ToSheet.Data3
        tabParts(num, nombreRelations, 5) = ""
```

```
            tabParts(num, nombreRelations, 6) = ""

        'Instance de la méta-classe de commentaires
        Case "CommentClass"
            'Renvoie le numéro de la classe de parts ou d'états associée au commentaire
            num = collectionFormes.Item(i).Connects.Item(1).ToSheet.Data3
            'Met à jour le compteur de relations de la classe
            tabParts(num, 0, 0) = tabParts(num, 0, 0) + 1
            'Renvoie le nombre de relations de la classe
            nombreRelations = tabParts(num, 0, 0)
            'Enregistre la relation dans le tableau
            tabParts(num, nombreRelations, 0) = collectionFormes.Item(i).Data1
            'Renvoie le nombre de caractères d'une chaîne (le commentaire)
            longueur = Len(collectionFormes.Item(i).Text)
            'Elimine les retours chariot du commentaire
            commentaire = collectionFormes.Item(i).Text
            For y = 1 To longueur
                If (Mid(commentaire, y, 1) Like Chr(10)) Then
                    commentaire = Mid(commentaire, 1, y - 1) & " " & Mid(commentaire, y + 1)
                End If
            Next y
            tabParts(num, nombreRelations, 1) = commentaire
            tabParts(num, nombreRelations, 2) = ""
            tabParts(num, nombreRelations, 3) = ""
            tabParts(num, nombreRelations, 4) = ""
            tabParts(num, nombreRelations, 5) = ""
            tabParts(num, nombreRelations, 6) = ""

        Case Else
            Debug.Print "La classe de parts, d'états ou d'attributs ne rentre pas " _
                & "dans la configuration prévue."

        End Select

    Next i

Next x

'3) Vérifie si le modèle graphique de "parts" est vide
If (tabParts(0, 0, 0) = 0) Then
    'Affiche le message à l'utilisateur
    message = "Un modèle graphique de ""parts"" doit obligatoirement contenir " _
    & "au moins une classe spécifique de parts." & Chr(10) & Chr(10)
    typeErreur = "Erreur : modèle vide"
    z = MsgBox(message, 0, typeErreur)
    'Force l'arrêt de la macro "traduireEnAlbertII"
    Exit Sub
End If
```

```
'4) Vérifie que toute les classes d'attributs différentes portent un nom différent
'Pour chaque classe de parts ou d'états ...
For x = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, x)
    '... pour chaque classe d'attributs ...
    For i = 1 To tabParts(num, 0, 0)
        'Teste si la classe d'attributs porte un nom (pas classe sous-ensemble, ni commentaire)
        If Not ((tabParts(num, i, 0) Like "isA*") Or _
            (tabParts(num, i, 0) Like "Comment")) Then
            '4.1) ... vérifie si une autre classe d'attributs (relations) ayant la même _
            classe d'origine et portant le même nom possède une classe destination _
            et/ou une cardinalité différentes ...
            'Pour toutes les classes d'attributs restantes ayant la même origine
            For y = i + 1 To tabParts(num, 0, 0)
                'Teste si le nom est identique
                If (tabParts(num, i, 1) Like tabParts(num, y, 1)) Then
                    'Teste si la cardinalité, la destination, la duplication ou _
                        l'ordre sont différents
                    If Not (tabParts(num, i, 2) Like tabParts(num, y, 2)) Or _
                        Not (tabParts(num, i, 3) Like tabParts(num, y, 3)) Or _
                        Not ((tabParts(num, i, 4) Like tabParts(num, y, 4)) Or _
                        (tabParts(tabParts(num, i, 4), 0, 2) Like _
                        tabParts(tabParts(num, y, 4), 0, 2))) Or _
                        Not (tabParts(num, i, 5) Like tabParts(num, y, 5)) Or _
                        Not (tabParts(num, i, 6) Like tabParts(num, y, 6)) Then
                        'Affiche le message à l'utilisateur
                        message = "Deux classes d'attributs différentes portent le même nom (""" _
                            & tabParts(num, i, 1) & """)" & Chr(10) & _
                            "et possèdent la même classe d'origine (""" _
                            & tabParts(num, 0, 2) & """) " & Chr(10) & _
                            "mais la cardinalité, la destination, la duplication ou " _
                            & "l'ordre est différent." & Chr(10) & _
                            "Or deux classes différentes ne peuvent pas porter le même nom." _
                            & Chr(10) & Chr(10) & _
                            "Veuillez corriger l'erreur avant de recommencer la traduction." _
                            & Chr(10) & Chr(10)
                        typeErreur = "Erreur : nom identique pour des classes d'attributs différentes"
                        z = MsgBox(message, 0, typeErreur)
                        'Force l'arrêt de la macro "traduireEnAlbertII"
                        Exit Sub
                    End If
                End If
            Next y
            '4.2) ... vérifie si une autre classe d'attributs (relations) portant _
            le même nom possède une classe d'origine différente.
            'Pour toutes les classes de parts ou d'états restantes
            For y = x + 1 To tabParts(0, 0, 0)
                'Pour toutes les classes d'attributs
                For w = 1 To tabParts(tabParts(0, 0, y), 0, 0)
```

```
                        'Teste si le nom de la classe est identique
                        If (tabParts(num, i, 1) Like tabParts(tabParts(0, 0, y), w, 1)) Then
                            'Affiche le message à l'utilisateur
                            message = "Deux classes d'attributs différentes portent le même nom (""" _
                                & tabParts(num, i, 1) & """) et possèdent" _
                                & Chr(10) & "deux classes d'origine différentes (""" _
                                & tabParts(num, 0, 2) & """ et """ & _
                                tabParts(tabParts(0, 0, y), 0, 2) & """)." & Chr(10) & _
                                "Or deux classes différentes ne peuvent pas porter le même nom." _
                                & Chr(10) & Chr(10) & _
                                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                                & Chr(10) & Chr(10)
                            typeErreur = "Erreur : nom identique pour des classes d'attributs différentes"
                            z = MsgBox(message, 0, typeErreur)
                            'Force l'arrêt de la macro "traduireEnAlbertII"
                            Exit Sub
                        End If
                    Next w
                Next y
            End If
        Next i
    Next x

    '5) Elimine les classes d'attributs qui apparaissent plusieurs fois dans le tableau
    'Pour chaque classe de parts ou d'états ...
    For x = 1 To tabParts(0, 0, 0)
        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, x)
        '... pour chaque classe d'attributs ...
        For i = 1 To tabParts(num, 0, 0)
            'Pour toutes les classes d'attributs restantes
            For y = i + 1 To tabParts(num, 0, 0)
                'Teste si les classes sont identiques : si le nom, la méta-classe/classe _
                spéciale et la destination sont identiques ou si le nom (non vide) est _
                identique (la seconde catégorie représente les classes d'attributs dont _
                la classe destination porte un nom qui est un type prédéfini AlbertII)
                If ((tabParts(num, i, 0) Like tabParts(num, y, 0)) And _
                    (tabParts(num, i, 1) Like tabParts(num, y, 1)) And _
                    (tabParts(num, i, 4) Like tabParts(num, y, 4))) Or _
                    ((tabParts(num, i, 1) Like tabParts(num, y, 1)) And _
                    Not (tabParts(num, i, 1) Like "")) Then
                    '... élimine la classe d'attributs identique.
                    'TEMPS 1 : fait avancer d'un rang les autres classes d'attributs
                    For w = y + 1 To tabParts(num, 0, 0)
                        tabParts(num, w - 1, 0) = tabParts(num, w, 0)
                        tabParts(num, w - 1, 1) = tabParts(num, w, 1)
                        tabParts(num, w - 1, 2) = tabParts(num, w, 2)
                        tabParts(num, w - 1, 3) = tabParts(num, w, 3)
                        tabParts(num, w - 1, 4) = tabParts(num, w, 4)
                        tabParts(num, w - 1, 5) = tabParts(num, w, 5)
```

```
                    tabParts(num, w - 1, 6) = tabParts(num, w, 6)
                Next w
                'TEMPS 2 : efface le dernier rang
                tabParts(num, tabParts(num, 0, 0), 0) = ""
                tabParts(num, tabParts(num, 0, 0), 1) = ""
                tabParts(num, tabParts(num, 0, 0), 2) = ""
                tabParts(num, tabParts(num, 0, 0), 3) = ""
                tabParts(num, tabParts(num, 0, 0), 4) = ""
                tabParts(num, tabParts(num, 0, 0), 5) = ""
                tabParts(num, tabParts(num, 0, 0), 6) = ""
                'TEMPS 3 : met le compteur de classes d'attributs à jour
                tabParts(num, 0, 0) = tabParts(num, 0, 0) - 1
                'TEMPS 4 : décrémente le compteur pour vérifier la classe d'attributs _
                        qui a pris la place de la classe d'attributs supprimée
                y = y - 1
            End If
        Next y
    Next i
Next x

'6) Vérifie qu'il n'existe qu'une seule relation entre deux mêmes classes. _
Les trois cas possibles où il pourrait y avoir deux relations de type différent (relation _
sous-ensemble d'une part et relation composant, contenu ou contenant d'autre part) est _
interdit car sinon une classe de parts pourrait être son propre composant/contenu ou une _
classe de conteneurs serait la destination d'une relation composant, contenu ou contenant.
'Pour chaque classe de parts ou d'états ...
For x = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, x)
    '... pour chaque classe d'attributs ...
    For i = 1 To tabParts(num, 0, 0)
        'Pour toutes les classes d'attributs restantes
        For y = i + 1 To tabParts(num, 0, 0)
            '... teste si les classes de destination sont identiques ... (commentaire !)
            If (tabParts(num, i, 4) Like tabParts(num, y, 4)) And _
                Not (tabParts(num, i, 4) Like "") Then
                '... et, dans ce cas, envoie un message à l'utilisateur.
                message = "Il ne peut exister qu'une seule relation entre deux classes." _
                    & Chr(10) & "Or les classes """ & tabParts(num, 0, 2) & """ et """ _
                    & tabParts(tabParts(num, i, 4), 0, 2) & _
                    """ ont en commun les relations """ & tabParts(num, i, 1) & _
                    """ et """ & tabParts(num, y, 1) & """." & Chr(10) & Chr(10) & _
                    "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : plusieurs relations entre deux mêmes classes"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                Exit Sub
            End If
        Next y
```

```
        Next i
Next x

'7) Vérifie l'absence de circuit ou de cycle au sein des relations de composition
For x = 1 To tabParts(0, 0, 0)
        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, x)
        'Teste si c'est une classe de parts composées ou de piles
        If (posséderRelation(num, "possible_components")) Then
                'Lance la vérification (voir "Module5")
                vérifierHiérarchie num, 1, 1, "classe de départ", "possible_components"
                'Teste la réussite ou l'échec de la vérification
                If Not (vérification) Then
                        'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
                        Exit Sub
                End If
        End If
Next x

'8) Vérifie l'absence de circuit ou de cycle au sein des relations de contenance
For x = 1 To tabParts(0, 0, 0)
        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, x)
        'Teste si c'est une classe de tampons
        If (posséderRelation(num, "possible_contents")) Then
                'Lance la vérification (voir "Module5")
                vérifierHiérarchie num, 1, 1, "classe de départ", "possible_contents"
                'Teste la réussite ou l'échec de la vérification
                If Not (vérification) Then
                        'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
                        Exit Sub
                End If
        End If
Next x

'9) Vérifie l'absence de cycles au sein des relations dispositifs physiques
For x = 1 To tabParts(0, 0, 0)
        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, x)
        'Teste si elle possède une relation dispositif physique
        If (posséderRelation(num, "possible_physical_features")) Then
                'Lance la vérification (voir "Module5")
                vérifierHiérarchie num, 1, 1, "classe de départ", "possible_physical_features"
                'Teste la réussite ou l'échec de la vérification
                If Not (vérification) Then
                        'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
                        Exit Sub
                End If
        End If
Next x
```

```
'10) Vérifie l'absence de cycles au sein des relations dispositifs géométriques
For x = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, x)
    'Teste si elle possède une relation dispositif géométriques
    If (posséderRelation(num, "possible_geometrical_features")) Then
        'Lance la vérification (voir "Module5")
        vérifierHiérarchie num, 1, 1, "classe de départ", "possible_geometrical_features"
        'Teste la réussite ou l'échec de la vérification
        If Not (vérification) Then
            'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
            Exit Sub
        End If
    End If
Next x

'11) Vérifie l'absence de cycles au sein des relations dispositifs de contiguïté
For x = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, x)
    'Teste si elle possède une relation dispositif de contiguïté
    If (posséderRelation(num, "contiguity_feature")) Then
        'Lance la vérification (voir "Module5")
        vérifierHiérarchie num, 1, 1, "classe de départ", "contiguity_feature"
        'Teste la réussite ou l'échec de la vérification
        If Not (vérification) Then
            'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
            Exit Sub
        End If
    End If
Next x

'12) Vérifie l'absence de cycles au sein des relations dispositifs d'enclos
For x = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, x)
    'Teste si elle possède une relation dispositif d'enclos
    If (posséderRelation(num, "enclosure_feature")) Then
        'Lance la vérification (voir "Module5")
        vérifierHiérarchie num, 1, 1, "classe de départ", "enclosure_feature"
        'Teste la réussite ou l'échec de la vérification
        If Not (vérification) Then
            'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
            Exit Sub
        End If
    End If
Next x
```

```
'13) Vérifie l'absence de cycles au sein des relations sous-ensembles ("isA")
For x = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, x)
    'Teste si elle possède une relation sous-ensemble
    If (posséderRelation(num, "isAOrigine")) Then
        'Lance la vérification (voir "Module5")
        vérifierHiérarchie num, 1, 1, "classe de départ", "isAOrigine"
        'Teste la réussite ou l'échec de la vérification
        If Not (vérification) Then
            'Force l'arrêt de l'exécution du code si la contrainte n'a pu être vérifiée
            Exit Sub
        End If
    End If
Next x

'14) Met à jour le tableau des parts composantes
'14.1) Complète les informations sur les sous-classes des parts composées
'Pour toutes les classes de parts composantes du tableau
For i = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste s'il s'agit d'une classe de parts composantes
    If Not (tabComposants(num, 0) = 0) Then
        'Lance la mise à jour du tableau des parts composantes
        mettreAJourTableauPartsComposantesPourComposées num
    End If
Next i
'14.2) Complète les informations sur les sous-classes des parts composantes
'Pour toutes les classes de parts composantes du tableau
For i = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste s'il s'agit d'une super classe et d'une classe de parts composantes
    If (posséderRelation(num, "isADestination") And Not (tabComposants(num, 0) = 0)) Then
        'Lance la mise à jour du tableau des parts composantes
        mettreAJourTableauPartsComposantesPourComposantes num
    End If
Next i

'15) Lance la vérification des contraintes sur les classes de parts et d'états ("Module3")
vérifierContraintesPartsEtats
'Teste la réussite ou l'échec de la vérification
If Not (vérification) Then
    'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
    Exit Sub
End If

'16) Vérifie que toutes les classes de parts composantes d'une classe de piles sont _
associées à au moins une classe de dispositifs de contiguïté qui définit cette classe de piles. _
```

```
Vérifie aussi le parallélisme entre les classe d'attributs composants et contiguïtés ayant _
la même classe de piles comme origine en tenant compte du phénomène de l'héritage.
'Pour toutes les classes de parts et d'états
For i = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste s'il s'agit d'une classe de piles feuille ou hors hiérarchie
    If (tabParts(num, 0, 1) Like "PileOfParts") And Not posséderRelation(num, "isADestination") Then
        '16.1) Construit le tableau temporaire contenant les numéros des classes de parts _
        composantes qui définissent les classes de dispositifs de contiguïté de la classe de piles. _
        Calcule en même temps la somme des cardinalités classes d'attributs contiguïtés.
        'Initialise les sommes des cardinalités des classes d'attributs contiguïtés
        borneContInf = 0
        borneContSup = 0
        'Initialise le compteur de classes de parts composantes.
        tabPile(0) = 0
        'Lance la construction du tableau contenant les classes de parts composantes
        construireTableauPartsComposantes num
        '16.2) Vérifie si les classes de parts composantes de la classe de piles _
        se trouvent dans le tableau "tabPile". _
        Calcule en même temps la somme des cardinalités des classes d'attributs composants
        'Initialise les sommes des cardinalités des classes d'attributs composants
        borneCompInf = 0
        borneCompSup = 0
        'Lance la vérification
        x = vérifierTableauPartsComposantes(num)
        If Not (x = 0) Then
            'Affiche le message à l'utilisateur
            message = "Toutes les classes de parts composantes d'une classe de piles doivent " _
                & "être associées à au moins une classe de dispositifs de contiguïté qui " _
                & "définit cette classe de piles." & Chr(10) & "Or la classe de parts composantes """ _
                & tabParts(x, 0, 2) & """ ne définit aucune classe de dispositifs de contiguïté " _
                & "de la classe de piles """ & tabParts(num, 0, 2) & """." & Chr(10) _
                & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : pile constituée de composants non contigus"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
            Exit Sub
        End If
        '16.3) Vérifie le parallélisme des classes d'attributs composants et contiguïtés. _
        La somme des bornes supérieures/inférieures des cardinalités des classes d'attributs _
        contiguïtés ne peut être inférieure de plus d'une unité par rapport à la somme _
        des bornes supérieures/inférieures des cardinalités des classes d'attributs composants.
        message = "La somme des bornes supérieures/inférieures des cardinalités des classes " _
            & "d'attributs contiguïtés associant une classe de piles à ses classes de " _
            & "dispositifs de contiguïté ne peut pas être inférieure de plus d'une unité par " _
            & "rapport à la somme des bornes supérieures/inférieures des cardinalités des " _
            & "classes d'attributs composants associant cette classe de piles à ses classes de " _
            & "parts composantes." & Chr(10) & Chr(10) _
```

```
            & "Or pour la classe de piles """ & tabParts(num, 0, 2) & """ :" & Chr(10)
        typeErreur = "Erreur : classes d'attributs contiguïtés et composants non parallèles"
        'Teste si composants indéfinis et contiguïtés entiers pour bornes supérieures
        If Not (IsNumeric(borneCompSup)) And IsNumeric(borneContSup) Then
            'Affiche le message à l'utilisateur
            message = message & "    la somme des bornes supérieures des relations " _
                & "contiguités est un entier : " & borneContSup & "," & Chr(10) _
                & "    la somme des bornes supérieures des relations " _
                & "composants est indéfinie : n." & Chr(10) & Chr(10) _
                & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
            Exit Sub
        End If
        'Teste si composants et contiguïtés sont des entiers pour bornes supérieures
        If (IsNumeric(borneCompSup) And IsNumeric(borneContSup)) Then
            'Teste si composants est trop élevée
            If (borneCompSup > borneContSup + 1) Then
                'Affiche le message à l'utilisateur
                message = message & "    la somme des bornes supérieures des relations " _
                    & "contiguités vaut : " & borneContSup & "," & Chr(10) & "    la somme " _
                    & "des bornes supérieures des relations " _
                    & "composants vaut : " & borneCompSup & "." & Chr(10) & Chr(10) _
                    & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
                Exit Sub
            End If
        End If
        'Teste si composants est trop élevée pour bornes inférieures
        If (Cint(borneCompInf) > Cint(borneContInf) + 1) Then
            'Affiche le message à l'utilisateur
            message = message & "    la somme des bornes inférieures des relations " _
                & "contiguités vaut : " & borneContInf & "," & Chr(10) _
                & "    la somme des bornes inférieures des relations " _
                & "composants vaut : " & borneCompInf & "." & Chr(10) & Chr(10) _
                & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
            Exit Sub
        End If
    End If
Next i
```

```
'17) Vérifie que toutes les classes de parts composantes d'une classe de piles sont _
associées à l'éventuelle classe de dispositifs d'enclos qui définit cette classe de piles. _
Vérifie aussi le parallélisme entre les classe d'attributs composants/contenants et dispositifs _
d'enclos ayant la même classe de parts comme destination en tenant compte de l'héritage.
'Pour toutes les classes de parts et d'états
For i = 1 To tabParts(0, 0, 0)
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste s'il s'agit d'une classe de piles ou de conteneurs feuille ou hors hiérarchie
    If ((tabParts(num, 0, 1) Like "PileOfParts") Or (tabParts(num, 0, 1) Like "Container")) And _
        Not posséderRelation(num, "isADestination") Then
        '17.1) Teste si la classe de piles FIFO ou LIFO ne possède pas de dispositifs d'enclos
        If (tabParts(num, 0, 1) Like "PileOfParts") And ((tabParts(num, 0, 3) Like "FIFO") _
            Or (tabParts(num, 0, 3) Like "LIFO")) And (compterRelation(num, "enclosure") = 0) Then
            'Affiche le message à l'utilisateur
            message = "La classe de piles " & tabParts(num, 0, 3) & " """ & tabParts(num, 0, 2) _
                & """ doit être associée à une classe de dispositifs d'enclos." & Chr(10) _
                & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : pile " & tabParts(num, 0, 3) & " sans dispositif d'enclos"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
            Exit Sub
        End If
        '17.2) Recherche l'éventuelle classe de dispositifs d'enclos
        numEnclos = chercherClasseDispositifsEnclos(num)
        'Teste si une classe de dispositifs d'enclos a été trouvée
        If Not (numEnclos = 0) Then
            '17.3) Construit le tableau temporaire contenant les numéros et les bornes _
            des classes d'attributs dispositifs d'enclos de la classe de dispositifs d'enclos
            'Enregistre le nom de la classe de piles ou de conteneurs
            tabEnclos(0, 1) = tabParts(num, 0, 2)
            'Enregistre les données sur la classe de piles ou de conteneurs
            tabEnclos(0, 2) = Switch(tabParts(num, 0, 1) Like "Container", "0", _
                tabParts(num, 0, 3) Like "standard", "2", True, "1")
            'Initialise le compteur du tableau
            tabEnclos(0, 0) = "0"
            'Complète le tableau avec les données sur les classes de dispositifs d'enclos
            construireTableauDispositifsEnclos numEnclos
            '17.4) Vérifie l'association des parts composantes et le parallélisme
            'Lance la vérification
            x = vérifierTableauDispositifsEnclos(num)
            'Teste la réussite ou l'échec de la vérification
            If Not (vérification) Then
                'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
                Exit Sub
            End If
            'Teste si la vérification a échoué pour la classe de piles
            If Not (x = 0) And Not (tabEnclos(0, 2) Like "0") Then
                'Affiche le message à l'utilisateur
```

```
                        message = "Toutes les classes de parts composantes d'une classe de piles " _
                            & "doivent être associées à la classe de dispositifs d'enclos qui définit " _
                            & "cette classe de piles." & Chr(10) & "Or la classe de parts composantes """ _
                            & tabParts(x, 0, 2) & """ ne définit pas la classe de dispositifs d'enclos " _
                            & "de la classe de piles """ & tabParts(num, 0, 2) & """." & Chr(10) _
                            & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                            & Chr(10) & Chr(10)
                        typeErreur = "Erreur : pile constituée de composants non enclos"
                        z = MsgBox(message, 0, typeErreur)
                        'Force l'arrêt de l'exécution du code si les contraintes n'ont pu être vérifiées
                        Exit Sub
                    End If
                End If
            End If
        Next i


        'DEUXIEME PARTIE : enregistrement des données dans un fichier en langage AlbertII

        'Lance la traduction du modèle graphique de "parts" ("Module4")
        enregistrerModèle
        'Teste la réussite ou l'échec de la traduction
        If Not (vérification) Then
            'Force l'arrêt de l'exécution du code si une erreur s'est produite dans l'accès au fichier
            Exit Sub
        End If

End Sub
```

```
'Met à jour le tableau des parts composantes en ajoutant à la classe de parts composantes "num"
'les sous-classes de ses classes de parts composées.
'num : numéro de la classe de parts composantes mise à jour.

Private Sub mettreAJourTableauPartsComposantesPourComposées(num As Integer)

    'Nombre initial de classes de parts composées de la classe de parts composantes
    Dim nombre As Integer
    'Compteur de classes de parts composées
    Dim i As Integer
    'Compteur de classes d'attributs d'une classe de parts composées
    Dim x As Integer
    'Compteur de classes de parts composées
    Dim y As Integer
    'Booléen indiquant si une classe de parts composées est déjà enregistrée dans le tableau
    Dim déjàEnregistré As Boolean

    'Renvoie le nombre de classes de parts composées de la classe de parts composantes
    nombre = tabComposants(num, 0)
```

```
    'Pour toutes les classes de parts composées de la classe de parts composantes
    For i = 1 To nombre
        'Pour toutes les classes d'attributs de la classe de parts composées
        For x = 1 To tabParts(tabComposants(num, i), 0, 0)
            'Teste s'il s'agit d'une classe d'attributs sous-classes
            If (tabParts(tabComposants(num, i), x, 0) Like "isADestination") Then
                'Initialise le booléen indiquant si la classe est déjà enregistrée
                déjàEnregistré = False
                'Teste si la classe de parts composées est déjà enregistrée dans le tableau
                For y = 1 To tabComposants(num, 0)
                    If (tabParts(tabComposants(num, i), x, 4) Like tabComposants(num, y)) Then
                        déjàEnregistré = True
                    End If
                Next y
                'Teste si la classe de parts composées n'est pas déjà enregistrée
                If Not (déjàEnregistré) Then
                    'Met à jour le compteur de classes de parts composées
                    tabComposants(num, 0) = tabComposants(num, 0) + 1
                    'Enregistre le numéro de la classe de parts composées
                    tabComposants(num, tabComposants(num, 0)) = tabParts(tabComposants(num, i), x, 4)
                End If
            End If
        Next x
    Next i
    'Teste si de nouvelles classes de parts composées ont été enregistrées
    If Not (nombre Like tabComposants(num, 0)) Then
        'Relance la mise à jour du tableau pour la même classe de parts composantes
        mettreAJourTableauPartsComposantesPourComposées num
    End If

End Sub
```

```
'Met à jour le tableau des parts composantes en ajoutant les classes de parts composées
'de la super classe de parts composantes "num" à ses sous-classes.
'num : numéro de la super classe de parts composantes dont les sous-classes seront mises à jour.

Private Sub mettreAJourTableauPartsComposantesPourComposantes(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Compteur de classes de parts composées de la super classe de parts composantes
    Dim x As Integer
    'Compteur de classes de parts composées
    Dim y As Integer
    'Booléen indiquant si une classe de parts composées est déjà enregistrée dans le tableau
    Dim déjàEnregistré As Boolean
    'Numéro de la sous-classe de parts composantes mise à jour
    Dim numéro As Integer
```

```
    'Pour toutes les classes d'attributs de la super classe de parts composantes
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs sous-classes
        If (tabParts(num, i, 0) Like "isADestination") Then
            'Enregistre les classes de parts composées dans la sous-classe
            For x = 1 To tabComposants(num, 0)
                'Initialise le booléen indiquant si la classe est déjà enregistrée
                déjàEnregistré = False
                'Teste si la classe de parts composées est déjà enregistrée dans le tableau
                For y = 1 To tabComposants(tabParts(num, i, 4), 0)
                    If (tabComposants(tabParts(num, i, 4), y) Like tabComposants(num, x)) Then
                        déjàEnregistré = True
                    End If
                Next y
                'Teste si la classe de parts composées n'est pas déjà enregistrée
                If Not (déjàEnregistré) Then
                    'Met à jour le compteur de classes de parts composées de la sous-classe
                    tabComposants(tabParts(num, i, 4), 0) = tabComposants(tabParts(num, i, 4), 0) + 1
                    'Enregistre le numéro de la classe de parts composées dans la sous-classe
                    tabComposants(tabParts(num, i, 4), tabComposants(tabParts(num, i, 4), 0)) _
                        = tabComposants(num, x)
                End If
            Next x
            'Lance la mise à jour du tableau pour les sous-classes de la sous-classe
            numéro = tabParts(num, i, 4)
            mettreAJourTableauPartsComposantesPourComposantes numéro
        End If
    Next i

End Sub
```

```
'Construit le tableau "tabPile" contenant les numéros des classes de parts composantes
'qui définissent les classes de dispositifs de contiguïté de la classe de piles "num"
'ou d'une sous-classe feuille de la classe de piles "num".
'Calcule également la somme des cardinalités des classes d'attributs contiguïtés.
'num : numéro de la classe de piles.

Private Sub construireTableauPartsComposantes(num As Integer)

    'Compteur de classes d'attributs de la classe de piles
    Dim i As Integer
    'Compteur de classes d'attributs d'une classe de dispositifs de contiguïté
    Dim x As Integer
    'Numéro d'une super classe de la classe de piles
    Dim numéro As Integer

    'Pour toutes les classes d'attributs de la classe de piles
    For i = 1 To tabParts(num, 0, 0)
```

```
        'Teste s'il s'agit d'une classe d'attributs contiguïtés
        If (tabParts(num, i, 0) Like "contiguity") Then
            'Additionne les bornes de la classe d'attributs contiguïtés
            borneContInf = Cint(borneContInf) + Cint(tabParts(num, i, 2))
            'Teste si la borne supérieure est un nombre indéfini
            If (tabParts(num, i, 3) Like "n") Or (borneContSup Like "n") Then
                borneContSup = "n"
            Else
                borneContSup = Cint(borneContSup) + Cint(tabParts(num, i, 3))
            End If
            'Pour toutes les classes d'attributs de la classe de dispositifs de contiguïté
            For x = 1 To tabParts(tabParts(num, i, 4), 0, 0)
                'Teste s'il s'agit d'une classe d'attributs dispositifs de contiguïté
                If (tabParts(tabParts(num, i, 4), x, 0) Like "contiguity_feature") Then
                    'Incrémente le compteur de classes de parts composantes du tableau
                    tabPile(0) = tabPile(0) + 1
                    'Enregistre la classe de parts composantes
                    tabPile(tabPile(0)) = tabParts(tabParts(num, i, 4), x, 4)
                End If
            Next x
        End If
        'Teste s'il s'agit d'une classe d'attributs sous-classes origine
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            'Lance la procédure avec la super classe de piles
            numéro = tabParts(num, i, 4)
            construireTableauPartsComposantes numéro
        End If
    Next i

End Sub
```

```
'Vérifie si les parts composantes de la classe de piles "num" se trouvent dans le tableau "tabPile"
'contenant les numéros des classes de parts composantes qui définissent les classes de dispositifs
'de contiguïté de la classe de piles "num" ou d'une super/sous-classe de la classe de piles "num".
'Calcule également la somme des cardinalités des classes d'attributs composants.
'num : numéro de la classe de piles.

Private Function vérifierTableauPartsComposantes(num As Integer)

    'Compteur de classes d'attributs de la classe de piles
    Dim i As Integer
    'Booléen indiquant que la classe de parts composantes se trouve dans le tableau _
    Valeur 0 : true _
    Valeur autre (> 0) : false (numéro de la classe absente du tableau)
    Dim contient As Integer
    'Numéro d'une super classe de la classe de piles
    Dim numéro As Integer
```

```
    'Pour toutes les classes d'attributs de la classe de piles
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs composants
        If (tabParts(num, i, 0) Like "possible_components") Then
            'Additionne les bornes de la classe d'attributs composants
            borneCompInf = Cint(borneCompInf) + Cint(tabParts(num, i, 2))
            'Teste si la borne supérieure est un nombre indéfini
            If (tabParts(num, i, 3) Like "n") Or (borneCompSup Like "n") Then
                borneCompSup = "n"
            Else
                borneCompSup = Cint(borneCompSup) + Cint(tabParts(num, i, 3))
            End If
            'Lance la recherche de la classe de parts composantes dans le tableau
            contient = chercherTableauPartsComposantes(tabParts(num, i, 4))
            'Teste si la classe de parts composantes n'est pas dans le tableau
            If Not (contient = 0) Then
                'Stoppe la vérification et renvoie le numéro de la classe absente
                vérifierTableauPartsComposantes = contient
                Exit Function
            End If
        End If
        'Teste s'il s'agit d'une classe d'attributs sous-classes origine
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            'Lance la procédure avec la super classe de piles
            numéro = tabParts(num, i, 4)
            contient = vérifierTableauPartsComposantes(numéro)
            If Not (contient = 0) Then
                'Stoppe la vérification et renvoie le numéro de la classe absente
                vérifierTableauPartsComposantes = contient
                Exit Function
            End If
        End If
    Next i
    vérifierTableauPartsComposantes = contient

End Function
```

```
'Vérifie si la classe de parts composantes "num" se trouve dans le tableau "tabPile"
'contenant les numéros des classes de parts composantes qui définissent les classes de dispositifs
'de contiguïté d'une classe de piles.
'num : numéro de la classe de parts composantes.

Private Function chercherTableauPartsComposantes(num As String)

    'Compteur de classes de parts composantes d'une classe de piles
    Dim i As Integer
    'Booléens indiquant que la classe de parts composantes se trouve dans le tableau _
    Valeur 0 : true _
    Valeur autre (> 0) : false (numéro de la classe absente du tableau)
```

```
    Dim contient As Integer, contientTemporaire As Integer
    'Numéro d'une super classe de la classe de piles
    Dim numéro As Integer

    'Initialise le booléen indiquant la présence de la classe de parts composantes
    contient = num
    'Pour toutes les classes de parts composantes de la classe de piles
    For i = 1 To tabPile(0)
        'Teste si la classe de parts composantes se trouve dans le tableau
        If (tabPile(i) Like num) Then
            chercherTableauPartsComposantes = 0
            Exit Function
        End If
    Next i
    'Vérifie que toutes les sous-classes de la classe de parts composantes se trouvent _
    dans le tableau.
    'Initialise le booléen temporaire
    contientTemporaire = contient
    'Pour toutes les classes d'attributs de la classe de parts composantes
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs sous-classe destination
        If (tabParts(num, i, 0) Like "isADestination") Then
            'Lance la procédure de recherche de la sous-classe dans le tableau
            contientTemporaire = chercherTableauPartsComposantes(tabParts(num, i, 4))
            'Teste si la sous-classe de la classe de parts composantes est absente du tableau
            If Not (contientTemporaire = 0) Then
                chercherTableauPartsComposantes = contient
                Exit Function
            End If
        End If
    Next i
    chercherTableauPartsComposantes = contientTemporaire

End Function
```

```
'Renvoie l'éventuelle classe de dispositifs d'enclos de la classe de piles ou de conteneurs "num".
'num : numéro de la classe de piles ou de conteneurs

Private Function chercherClasseDispositifsEnclos(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Numéro de la classe de dispositifs d'Enclos
    Dim numEnclos As Integer
    'Numéro d'une super classe de la classe de piles ou de conteneurs
    Dim numéro As Integer

    'Initialise le numéro de la classe de dispositifs d'enclos
    numEnclos = 0
```

```
    'Pour toutes les classes d'attributs de la classe de piles ou de conteneurs
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs enclos
        If (tabParts(num, i, 0) Like "enclosure") Then
            numEnclos = tabParts(num, i, 4)
        End If
    Next i
    'Teste si la classe de dispositifs d'enclos n'a pas été trouvée
    If (numEnclos = 0) Then
        'Teste si la première classe d'attributs est une classe d'attributs sous-classe origine
        If (tabParts(num, 1, 0) Like "isAOrigine") Then
            'Relanche la recherche pour la super classe de la classe de piles ou de conteneurs
            numéro = tabParts(num, 1, 4)
            numEnclos = chercherClasseDispositifsEnclos(numéro)
        End If
    End If
    chercherClasseDispositifsEnclos = numEnclos

End Function
```

```
'Enregistre les numéros et les bornes des classes d'attributs dispositifs d'enclos de
'la classe de dispositifs d'enclos "num" dans le tableau tabEnclos
'num : numéro de la classe de dispositifs d'enclos

Private Sub construireTableauDispositifsEnclos(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer

    'Pour toutes les classes d'attributs de la classe de dispositifs d'enclos
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs dispositifs d'enclos
        If (tabParts(num, i, 0) Like "enclosure_feature") Then
            'Incrémente le compteur de classes d'attributs dispositifs d'enclos
            tabEnclos(0, 0) = tabEnclos(0, 0) + 1
            'Enregistre le numéro de la classe de parts composantes/contenues
            tabEnclos(tabEnclos(0, 0), 0) = tabParts(num, i, 4)
            'Enregistre les bornes inférieure et supérieure de la classe d'attributs
            tabEnclos(tabEnclos(0, 0), 1) = tabParts(num, i, 2)
            tabEnclos(tabEnclos(0, 0), 2) = tabParts(num, i, 3)
        End If
    Next i

End Sub
```

```
'Vérifie si les parts composantes de la classe de piles "num" se trouvent dans le tableau "tabEnclos"
'contenant les numéros des classes de parts composantes qui définissent les classes de dispositifs
'd'enclos de la classe de piles "num" ou d'une super/sous-classe de la classe de piles "num".
'Vérifie également le parallélisme entre les cardinalités des classes d'attributs composants
'ou contenants et dispositifs d'enclos de la classe de piles ou de conteneurs "num".
'num : numéro de la classe de piles ou de conteneurs.
Private Function vérifierTableauDispositifsEnclos(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Booléen indiquant que la classe de parts composantes/contenues se trouve dans le tableau _
    Valeur 0 : true _
    Valeur autre (> 0) : false (numéro de la classe absente du tableau)
    Dim contient As Integer
    'Numéro d'une super classe de la classe de piles ou de conteneurs
    Dim numéro As Integer

    'Pour toutes les classes d'attributs de la classe de piles ou de conteneurs
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs composants ou contenants
        If (tabParts(num, i, 0) Like "possible_components") Or _
            (tabParts(num, i, 0) Like "contain") Then
            'Lance la recherche de la classe de parts composantes/contenues dans le tableau
            contient = chercherTableauDispositifsEnclos(tabParts(num, i, 4), _
                tabParts(num, i, 2), tabParts(num, i, 3))
            'Teste si la classe de parts composantes/contenues n'est pas dans le tableau _
            et s'il s'agit d'une classe de piles
            If (Not (contient = 0) And Not (tabEnclos(0, 2) Like "0")) Or _
                (vérification = False) Then
                'Stoppe la vérification et renvoie le numéro de la classe absente
                vérifierTableauDispositifsEnclos = contient
                Exit Function
            End If
        End If
        'Teste s'il s'agit d'une classe d'attributs sous-classes origine
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            'Lance la procédure avec la super classe de piles
            numéro = tabParts(num, i, 4)
            contient = vérifierTableauDispositifsEnclos(numéro)
            'Teste si la classe de parts composantes/contenues n'est pas dans le tableau _
            et s'il s'agit d'une classe de piles
            If Not (contient = 0) And Not (tabEnclos(0, 2) Like "0") Then
                'Stoppe la vérification et renvoie le numéro de la classe absente
                vérifierTableauDispositifsEnclos = contient
                Exit Function
            End If
        End If
    Next i
    vérifierTableauDispositifsEnclos = contient
End Function
```

```
'Vérifie si la classe de parts composantes "num" se trouve dans le tableau "tabEnclos"
'contenant les numéros des classes de parts composantes qui définissent les classes de dispositifs
'd'enclos de la classe de piles "num" ou d'une super/sous-classe de la classe de piles "num".
'Vérifie également le parallélisme entre les cardinalités de la classe d'attributs composants
'ou contenants et dispositifs d'enclos de la classe de parts composantes ou contenues "num".
'num : numéro de la classe de parts composantes ou contenues.
'borneInf : borne inférieure de la classe d'attributs composants ou contenants
'borneSup : borne supérieure de la classe d'attributs composants ou contenants

Private Function chercherTableauDispositifsEnclos(num As String, borneInf As String, borneSup As String)

    'Compteur de classes de parts composantes/contenues d'une classe de piles ou de conteneurs
    Dim i As Integer
    'Booléens indiquant que la classe de parts composantes/contenues se trouve dans le tableau _
    Valeur 0 : true _
    Valeur autre (> 0) : false (numéro de la classe absente du tableau)
    Dim contient As Integer, contientTemporaire As Integer
    'Numéro d'une super classe de la classe de piles ou de conteneurs
    Dim numéro As Integer
    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer

    'Initialise le booléen indiquant la présence de la classe de parts composantes/contenues
    contient = num
    'Pour toutes les classes de parts composantes/contenues de la classe de piles/conteneurs
    For i = 1 To tabEnclos(0, 0)
        'Teste si la classe de parts composantes/contenues se trouve dans le tableau
        If (tabEnclos(i, 0) Like num) Then
            'Teste s'il s'agit d'une classe de conteneurs
            If (tabEnclos(0, 2) Like "0") Then
                message = "Les bornes supérieure/inférieure de la cardinalité d'une classe d'attributs " _
                    & "dispositifs d'enclos associant une classe de dispositifs d'enclos à une classe " _
                    & "de parts contenues ne peuvent pas être supérieures aux bornes supérieure/" _
                    & "inférieure de la cardinalité de la classe d'attributs contenants associant la " _
                    & "classe de conteneurs de cette classe de dispositifs d'enclos à cette classe de " _
                    & "parts contenues." & Chr(10) & Chr(10) _
                    & "Or pour la classe de conteneurs """ & tabEnclos(0, 1) & """ associée à la classe " _
                    & "de parts contenues """ & tabParts(num, 0, 2) & """ comme destination des " _
                    & "classes d'attributs :" & Chr(10)
                typeErreur = "Erreur : classes d'attributs dispositifs d'enclos et contenants non parallèles"
                'Teste si la borne inférieure de la relation dispositifs d'enclos est supérieure
                If (Cint(tabEnclos(i, 1)) > Cint(borneInf)) Then
                    'Affiche le message à l'utilisateur
                    message = message & "    la borne inférieure de la relation dispositif d'enclos " _
                        & "vaut : " & tabEnclos(i, 1) & "," & Chr(10) _
                        & "    la borne inférieure de la relation contenant " _
                        & "vaut : " & borneInf & Chr(10) & Chr(10) _
```

```
                & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Function
        End If
        'Teste si dispositifs d'enclos indéfinis et contenants entiers pour bornes supérieures
        If Not (IsNumeric(tabEnclos(i, 2))) And IsNumeric(borneSup) Then
            'Affiche le message à l'utilisateur
            message = message & "    la borne supérieure de la relation dispositif d'enclos " _
                & "est indéfinie : " & tabEnclos(i, 2) & "," & Chr(10) _
                & "    la borne supérieure de la relation contenant " _
                & "vaut : " & borneSup & Chr(10) & Chr(10) _
                & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Function
        End If
        'Teste si la borne supérieure de la relation dispositif d'enclos est supérieure
        If IsNumeric(tabEnclos(i, 2)) And IsNumeric(borneSup) Then
            If (Cint(tabEnclos(i, 2)) > Cint(borneSup)) Then
                'Affiche le message à l'utilisateur
                message = message & "    la borne supérieure de la relation dispositif " _
                    & "d'enclos vaut : " & tabEnclos(i, 2) & "," & Chr(10) _
                    & "    la borne supérieure de la relation contenant " _
                    & "vaut : " & borneSup & Chr(10) & Chr(10) _
                    & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Function
            End If
        End If
    Else
    'Il s'agit d'une classe de piles
        message = "Les bornes supérieure/inférieure de la cardinalité d'une classe " _
            & "d'attributs dispositifs d'enclos associant une classe de dispositifs d'enclos à " _
            & "une classe de parts composantes doivent être identiques aux bornes supérieure/" _
            & "inférieure de la cardinalité de la classe d'attributs composants associant la " _
            & "classe de piles de cette classe de dispositifs d'enclos à cette classe de parts " _
            & "composantes." & Chr(10) _
            & "Toutefois, les bornes des classes de piles FIFO-LIFO ou standards peuvent " _
            & "être inférieures respectivement d'une ou de deux unités." & Chr(10) & Chr(10) _
            & "Or pour la classe de piles """ & tabEnclos(0, 1) & """ associée à la classe " _
            & "de parts composantes """ & tabParts(num, 0, 2) & """ comme destination " _
            & "des classes d'attributs :" & Chr(10)
```

```
            typeErreur = "Erreur : classes d'attributs dispositifs d'enclos et composants non parallèles"
            'Teste si les bornes inférieures ne sont pas identiques
            If (Cint(tabEnclos(i, 1)) > Cint(borneInf)) Or _
                (Cint(tabEnclos(i, 1)) < Cint(borneInf) - Cint(tabEnclos(0, 2))) Then
                'Affiche le message à l'utilisateur
                message = message & "    la borne inférieure de la relation dispositif d'enclos " _
                    & "vaut : " & tabEnclos(i, 1) & "," & Chr(10) _
                    & "    la borne inférieure de la relation composant " _
                    & "vaut : " & borneInf & Chr(10) & Chr(10) _
                    & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Function
            End If
            'Teste si dispositifs d'enclos ou composants indéfinis pour bornes supérieures
            If IsNumeric(tabEnclos(i, 2)) Xor IsNumeric(borneSup) Then
                'Affiche le message à l'utilisateur
                message = message & "    la borne supérieure de la relation dispositif d'enclos " _
                    & "vaut : " & tabEnclos(i, 2) & "," & Chr(10) _
                    & "    la borne supérieure de la relation composant " _
                    & "vaut : " & borneSup & Chr(10) & Chr(10) _
                    & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Function
            End If
            'Teste si les bornes supérieures ne sont pas identiques
            If IsNumeric(tabEnclos(i, 2)) And IsNumeric(borneSup) Then
                If (Cint(tabEnclos(i, 2)) > Cint(borneSup)) Or _
                    (Cint(tabEnclos(i, 2)) < Cint(borneSup) - Cint(tabEnclos(0, 2))) Then
                    message = message & "    la borne supérieure de la relation dispositif " _
                        & "d'enclos vaut : " & tabEnclos(i, 2) & "," & Chr(10) _
                        & "    la borne supérieure de la relation composant " _
                        & "vaut : " & borneSup & Chr(10) & Chr(10) _
                        & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                        & Chr(10) & Chr(10)
                    z = MsgBox(message, 0, typeErreur)
                    'Force l'arrêt de la macro "traduireEnAlbertII"
                    vérification = False
                    Exit Function
                End If
            End If
        End If
        chercherTableauDispositifsEnclos = 0
        Exit Function
    End If
```

```
    Next i
    'Vérifie que toutes les sous-classes de la classe de parts composantes/contenues se trouvent _
    dans le tableau.
    'Initialise le booléen temporaire
    contientTemporaire = contient
    'Pour toutes les classes d'attributs de la classe de parts composantes/contenues
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs sous-classe destination
        If (tabParts(num, i, 0) Like "isADestination") Then
            'Lance la procédure de recherche de la sous-classe dans le tableau
            contientTemporaire = chercherTableauDispositifsEnclos(tabParts(num, i, 4), _
                borneInf, borneSup)
            'Teste si la sous-classe de la classe de parts composantes est absente du tableau _
            et s'il s'agit d'une classe de piles
            If (Not (contientTemporaire = 0) And Not (tabEnclos(0, 2) Like "0")) Or _
                (vérification = False) Then
                chercherTableauDispositifsEnclos = contient
                Exit Function
            End If
        End If
    Next i
    chercherTableauDispositifsEnclos = contientTemporaire

End Function
```

```
'Renvoie la valeur "VRAI" si la classe de parts ou d'états "num" possède
'la relation "relation", renvoie "FAUX" sinon.
'num : numéro de la classe de parts ou d'états dont les relations sont testées.
'relation : type de relation (classe d'attributs).

Public Function posséderRelation(num As Integer, relation As String)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Booléen indiquant si la relation "relation" existe
    Dim existenceRelation As Boolean

    'Initialise le booléen
    existenceRelation = False
    'Pour toutes les classes d'attributs de la classe de parts ou d'états
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit de la relation recherchée
        If (tabParts(num, i, 0) Like relation) Then
            existenceRelation = True
        End If
    Next i
    posséderRelation = existenceRelation

End Function
```

F.5. Module standard "Module2"

```
'Module nommé "Module2". _
Teste si un modèle graphique de "parts" vérifie les contraintes sur les classes d'attributs. _
La correction du modèle graphique est testée lors de sa traduction en langage AlbertII : _
les trois procédures de ce module sont appelées par la macro "traduireEnAlbertII" ("Module1").

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text

'Tableau dynamique qui contient des informations sur les classes d'états _
qui sont la destination d'une seule classe d'attributs (relation). _
L'indice correspond au numéro de la classe au sein du modèle graphique. _
La valeur (string) indique le nom de la relation dont la classe est la destination.
Public tabRelations() As String
```

```
'Initialise le tableau des relations et une variable au début de la procédure de vérification
'taille : taille du tableau dynamique

Public Sub initialiserVérifierContraintes(taille As Integer)

    'Indice du tableau des relations
    Dim i As Integer

    'Réattribue de l'espace de stockage à la variable de tableau dynamique
    ReDim tabRelations(0 To taille) As String
    'Initialise les cellules du tableau
    For i = 0 To taille
        tabRelations(i) = ""
    Next i
    'Initialise la variable booléenne appartenant au "Module1"
    vérification = True

End Sub
```

```
'Vérifie les contraintes sur les classes d'attributs (origine, destination et cardinalité)
'forme : forme représentant la classe d'attributs (relation) à vérifier

Public Sub vérifierContraintesAttributs(forme As Visio.Shape)

    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer
```

```
'PREMIERE PARTIE : vérifie si la relation possède une forme à chaque extrémité

If Not (forme.Connects.Count = 2) Then
    'Affiche le message à l'utilisateur
    'Teste si la relation porte un nom (la relation sous-ensemble n'en porte pas)
    If (forme.Layer(1).Name Like "SubClass") Then
        message = "Une relation sous-ensemble" & "  (page n° " & forme.ContainingPage & _
            ")  ne possède pas une classe de parts à chaque extrémité." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
    Else
        message = "La relation " & forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & _
            ")  ne possède pas une classe de parts ou d'états à chaque extrémité." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
    End If
    typeErreur = "Erreur : relation sans extrémité"
    z = MsgBox(message, 0, typeErreur)
    'Force l'arrêt de la macro "traduireEnAlbertII"
    vérification = False
    Exit Sub
End If

'DEUXIEME PARTIE : vérifie si les classes aux extrémités de la relation, _
                  si la syntaxe de la cardinalité de la relation et _
                  si le nombre de types de relations par classe est correct.

'Renvoie le nom de la méta-classe dont la classe d'attributs est l'instance _
ou le nom de la classe spéciale dont la classe d'attributs est le sous-ensemble
Select Case forme.Data1

'Classe d'attributs composants
Case "possible_components"
    '1) Vérifie si l'origine est une classe de piles ou de parts composées
    If Not ((forme.Connects.Item(1).ToSheet.Data1 Like "CompoundPartClass") Or _
        (forme.Connects.Item(1).ToSheet.Data1 Like "PileOfParts")) Then
        'Affiche le message à l'utilisateur
        message = "L'origine de la relation composant " & _
            forme.Cells("Prop.Nom").Formula & "  (page n° " & forme.ContainingPage _
            & ")  doit être une classe de piles ou de parts composées." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : origine de la relation composant"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
```

```
        '2) Vérifie si la destination est une classe de parts (sauf tampons et conteneurs)
        If Not ((forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Buffer") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation composant " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & _
                ")  doit être une classe de parts (sauf tampons et conteneurs)." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation composant"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie si la relation n'associe pas une classe à elle-même
        If (forme.Connects.Item(1).ToSheet.Data3 Like _
            forme.Connects.Item(2).ToSheet.Data3) Then
            'Affiche le message à l'utilisateur
            message = "La relation composant " & forme.Cells("Prop.Nom").Formula & _
                "  (page n° " & forme.ContainingPage & _
                ")  associe la classe de piles ou de parts composées " & _
                forme.Connects.Item(1).ToSheet.Cells("Prop.Nom").Formula & " à elle-même." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : classe de parts associée à elle-même"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '4) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If

    'Classe d'attributs contenus
    Case "possible_contents"
        '1) Vérifie si l'origine est une classe de tampons
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "Buffer") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation contenu " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & forme.ContainingPage _
                & ")  doit être une classe de tampons." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation contenu"
```

```
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe de parts (sauf conteneurs)
        If Not ((forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation contenu " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & _
                ")  doit être une classe de parts (sauf conteneurs)." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation contenu"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie si la relation n'associe pas une classe à elle-même
        If (forme.Connects.Item(1).ToSheet.Data3 Like _
            forme.Connects.Item(2).ToSheet.Data3) Then
            'Affiche le message à l'utilisateur
            message = "La relation contenu " & forme.Cells("Prop.Nom").Formula & _
                "  (page n° " & forme.ContainingPage & _
                ")  associe la classe de tampons " & _
                forme.Connects.Item(1).ToSheet.Cells("Prop.Nom").Formula & " à elle-même." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : classe de tampons associée à elle-même"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '4) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If

    'Classe d'attributs contenants
    Case "contain"
        '1) Vérifie si l'origine est une classe de conteneurs
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "Container") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation contenant " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & forme.ContainingPage _
```

```
                & ")  doit être une classe de conteneurs." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation contenant"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe de parts (sauf conteneurs)
        If Not ((forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation contenant " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & _
                ")  doit être une classe de parts (sauf conteneurs)." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation contenant"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If

'Classe d'attributs identités
Case "possible_identities"
        '1) Vérifie si l'origine est une classe de parts
        If Not (forme.Connects.Item(1).ToSheet.Layer(1).Name Like "PartClass") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation identité " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de parts." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation identité"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe d'identités
        If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartIdentityClass") Then
            'Affiche le message à l'utilisateur
```

```
            message = "La destination de la relation identité " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe d'identités." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation identité"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie si une classe d'identités est la destination de plusieurs relations _
        identités, ce qui serait contraire aux contraintes sur les classes d'identités
        If (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like "") Then
            tabRelations(forme.Connects.Item(2).ToSheet.Data3) = forme.Cells("Prop.Nom").ResultStr(0)
        Else
            'Teste s'il s'agit de deux relations différentes (noms différents)
            If Not (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like _
                forme.Cells("Prop.Nom").ResultStr(0)) Then
                'Affiche le message à l'utilisateur
                message = "Une classe d'identités ne peut pas être la destination " _
                    & "de plusieurs relations identités." _
                    & Chr(10) & "C'est pourtant le cas de la classe d'identités " _
                    & forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula & _
                    "  (page n° " & forme.ContainingPage & ")." _
                    & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : classe d'identités multi-destination"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Sub
            End If
        End If

    'Classe d'attributs positions
    Case "possible_positions"
        '1) Vérifie si l'origine est une classe de parts
        If Not (forme.Connects.Item(1).ToSheet.Layer(1).Name Like "PartClass") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation position " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de parts." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation position"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
```

```
        End If
        '2) Vérifie si la destination est une classe de positions
        If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartPositionClass") Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation position " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de positions." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation position"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
         '3) Vérifie si une classe de positions est la destination de plusieurs relations _
         positions, ce qui serait contraire aux contraintes sur les classes de positions
        If (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like "") Then
            tabRelations(forme.Connects.Item(2).ToSheet.Data3) = forme.Cells("Prop.Nom").ResultStr(0)
        Else
            'Teste s'il s'agit de deux relations différentes (noms différents)
            If Not (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like _
                forme.Cells("Prop.Nom").ResultStr(0)) Then
                'Affiche le message à l'utilisateur
                message = "Une classe de positions ne peut pas être la destination " _
                    & "de plusieurs relations positions." _
                    & Chr(10) & "C'est pourtant le cas de la classe de positions " _
                    & forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula & _
                    "  (page n° " & forme.ContainingPage & ")." _
                    & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : classe de positions multi-destination"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Sub
            End If
        End If

    'Classe d'attributs dispositifs physiques
    Case "possible_physical_features"
        '1) Vérifie si l'origine est une classe de parts
        If Not (forme.Connects.Item(1).ToSheet.Layer(1).Name Like "PartClass") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation dispositif physique " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de parts." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation dispositif physique"
```

```
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe de dispositifs physiques
        If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartPhysicalFeatureClass") Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation dispositif physique " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs physiques." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation dispositif physique"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If
        '4) Indique qu'une classe de dispositifs physiques est la destination _
        d'une relation dispositif physique
        tabRelations(forme.Connects.Item(2).ToSheet.Data3) = True

    'Classe d'attributs géométries
    Case "possible_geometry"
        '1) Vérifie si l'origine est une classe de parts composées
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "CompoundPartClass") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation géométrie " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de parts composées." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation géométrie"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination une classe de dispositifs géométriques
        If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartGeometricalFeatureClass") Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation géométrie " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs géométriques." _
```

```
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation géométrie"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If
        '4) Vérifie si une classe de dispositifs géométriques est la destination _
        de plusieurs relations géométries, ce qui serait contraire aux contraintes _
        sur les classes de dispositifs géométriques
        If (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like "") Then
            tabRelations(forme.Connects.Item(2).ToSheet.Data3) = forme.Cells("Prop.Nom").ResultStr(0)
        Else
            'Teste s'il s'agit de deux relations différentes (noms différents)
            If Not (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like _
                forme.Cells("Prop.Nom").ResultStr(0)) Then
                'Affiche le message à l'utilisateur
                message = "Une classe de dispositifs géométriques ne peut pas être " _
                    & "la destination de plusieurs relations géométries." _
                    & Chr(10) & "C'est pourtant le cas de la classe de dispositifs géométriques " _
                    & forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula & _
                    "  (page n° " & forme.ContainingPage & ")." _
                    & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : classe de dispositifs géométriques multi-destination"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Sub
            End If
        End If

    'Classe d'attributs dispositifs géométriques
    Case "possible_geometrical_features"
        '1) Vérifie si l'origine est une classe de dispositifs géométriques
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "PartGeometricalFeatureClass") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation dispositif géométrique " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs géométriques." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation dispositif géométrique"
            z = MsgBox(message, 0, typeErreur)
```

```
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe de parts composantes
        'Teste si la destination est une classe de parts (sauf tampons et conteneurs)
        If Not ((forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Buffer") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation dispositif géométrique " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & forme.ContainingPage _
                & ")  doit être une classe de parts (sauf tampons et conteneurs)." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation dispositif géométrique"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If

    'Classe d'attributs dispositifs de fixation
    Case "possible_fixing_features"
        '1) Vérifie si l'origine est une classe de dispositifs de fixation
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "PartFixingFeatureClass") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation dispositif de fixation " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs de fixation." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation dispositif de fixation"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe de dispositifs géométriques
        If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartGeometricalFeatureClass") Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation dispositif de fixation " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs géométriques." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
```

```
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation dispositif de fixation"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If

'Classe d'attributs dispositifs de contiguïté
Case "contiguity_feature"
        '1) Vérifie si l'origine est une classe de dispositifs de contiguïté
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "PartContiguityFeature") Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation dispositif de contiguïté " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs de contiguïté." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation dispositif de contiguïté"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la destination est une classe de parts composantes
        'Teste si la destination est une classe de parts (sauf conteneurs)
        If Not ((forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation dispositif de contiguïté " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de parts (sauf conteneurs)." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation dispositif de contiguïté"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If

'Classe d'attributs dispositifs d'enclos
Case "enclosure_feature"
        '1) Vérifie si l'origine est une classe de dispositifs d'enclos
        If Not (forme.Connects.Item(1).ToSheet.Data1 Like "PartEnclosureFeature") Then
```

```
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation dispositif d'enclos " & _
                  forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                  forme.ContainingPage & ")  doit être une classe de dispositifs d'enclos." _
                  & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                  & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation dispositif d'enclos"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
      End If
      '2) Vérifie si la destination est une classe de parts composantes
      'Teste si la destination est une classe de parts (sauf conteneurs)
      If Not ((forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") And _
            Not (forme.Connects.Item(2).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation dispositif d'enclos " & _
                  forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                  forme.ContainingPage & ")  doit être une classe de parts (sauf conteneurs)." _
                  & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                  & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation dispositif d'enclos"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
      End If
      '3) Vérifie la syntaxe de la cardinalité
      If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
      End If

'Classe d'attributs contiguïtés
Case "contiguity"
      '1) Vérifie si l'origine est une classe de piles ou de conteneurs
      If Not ((forme.Connects.Item(1).ToSheet.Data1 Like "PileOfParts") Or _
            (forme.Connects.Item(1).ToSheet.Data1 Like "Container")) Then
            'Affiche le message à l'utilisateur
            message = "L'origine de la relation contiguïté " & _
                  forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                  forme.ContainingPage & ")  doit être une classe de piles ou de conteneurs." _
                  & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                  & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation contiguïté"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
```

```
        End If
        '2) Vérifie si la destination est une classe de dispositifs de contiguïté
        If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartContiguityFeature") Then
            'Affiche le message à l'utilisateur
            message = "La destination de la relation contiguïté " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  doit être une classe de dispositifs de contiguïté." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : destination de la relation contiguïté"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie la syntaxe de la cardinalité
        If Not (vérifierCardinalité(forme)) Then
            vérification = False
            Exit Sub
        End If
        '4) Vérifie si une classe de dispositifs de contiguïté est la destination _
        de plusieurs relations contiguïtés, ce qui serait contraire aux contraintes _
        sur les classes de dispositifs de contiguïté
        If (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like "") Then
            tabRelations(forme.Connects.Item(2).ToSheet.Data3) = forme.Cells("Prop.Nom").ResultStr(0)
        Else
            'Teste s'il s'agit de deux relations différentes (noms différents)
            If Not (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like _
                forme.Cells("Prop.Nom").ResultStr(0)) Then
                'Affiche le message à l'utilisateur
                message = "Une classe de dispositifs de contiguïté ne peut pas être la destination " _
                    & "de plusieurs relations contiguïtés." & Chr(10) & _
                    "C'est pourtant le cas de la classe de dispositifs de contiguïté " _
                    & forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula & _
                    "  (page n° " & forme.ContainingPage & ")." _
                    & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : classe de dispositifs de contiguïté multi-destination"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Sub
            End If
        End If

    'Classe d'attributs enclos
    Case "enclosure"
        '1) Vérifie si l'origine est une classe de piles ou de conteneurs
        If Not ((forme.Connects.Item(1).ToSheet.Data1 Like "PileOfParts") Or _
            (forme.Connects.Item(1).ToSheet.Data1 Like "Container")) Then
```

```
        'Affiche le message à l'utilisateur
        message = "L'origine de la relation enclos " & _
            forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  doit être une classe de piles ou de conteneurs." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : origine de la relation enclos"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '2) Vérifie si la destination est une classe de dispositifs d'enclos
    If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartEnclosureFeature") Then
        'Affiche le message à l'utilisateur
        message = "La destination de la relation enclos " & _
            forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  doit être une classe de dispositifs d'enclos." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : destination de la relation enclos"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '3) Vérifie si une classe de dispositifs d'enclos est la destination de plusieurs relations _
    enclos, ce qui serait contraire aux contraintes sur les classes de dispositifs d'enclos
    If (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like "") Then
        tabRelations(forme.Connects.Item(2).ToSheet.Data3) = forme.Cells("Prop.Nom").ResultStr(0)
    Else
        'Teste s'il s'agit de deux relations différentes (noms différents)
        If Not (tabRelations(forme.Connects.Item(2).ToSheet.Data3) Like _
            forme.Cells("Prop.Nom").ResultStr(0)) Then
            'Affiche le message à l'utilisateur
            message = "Une classe de dispositifs d'enclos ne peut pas être la destination " _
                & "de plusieurs relations enclos." & Chr(10) & _
                "C'est pourtant le cas de la classe de dispositifs d'enclos " _
                & forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula & _
                "  (page n° " & forme.ContainingPage & ")." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : classe de dispositifs d'enclos multi-destination"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
```

```
'Classe d'attributs valeurs
Case "possible_values"
    '1) Vérifie si l'origine est une classe de dispositifs
    If Not (forme.Connects.Item(1).ToSheet.Layer(1).Name Like "PartFeatureClass") Then
        'Affiche le message à l'utilisateur
        message = "L'origine de la relation valeur " & _
            forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  doit être une classe de dispositifs." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : origine de la relation valeur"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '2) Vérifie si la destination est une classe de dispositifs de valeur
    If Not (forme.Connects.Item(2).ToSheet.Data1 Like "PartFeatureValueClass") Then
        'Affiche le message à l'utilisateur
        message = "La destination de la relation valeur " & _
            forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  doit être une classe de dispositifs de valeur." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : destination de la relation valeur"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '3) Vérifie la syntaxe de la cardinalité
    If Not (vérifierCardinalité(forme)) Then
        vérification = False
        Exit Sub
    End If
     '4) Indique qu'une classe de dispositifs de valeur est la destination _
     d'une relation valeur
    tabRelations(forme.Connects.Item(2).ToSheet.Data3) = True

'Classe d'attributs sous-classes
Case "isA"
    '1) Vérifie si l'origine est une classe de parts
    If Not (forme.Connects.Item(1).ToSheet.Layer(1).Name Like "PartClass") Then
        'Affiche le message à l'utilisateur
        message = "L'origine d'une relation sous-ensemble doit être une classe de parts." _
            & Chr(10) & "Or ce n'est pas le cas de la relation sous-ensemble qui a pour " _
            & "origine la classe " & forme.Connects.Item(1).ToSheet.Cells("Prop.Nom").Formula _
            & "  (page n° " & forme.ContainingPage & ")." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
```

```
        typeErreur = "Erreur : origine de la relation sous-ensemble"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '2) Vérifie si la destination est une classe de parts
    If Not (forme.Connects.Item(2).ToSheet.Layer(1).Name Like "PartClass") Then
        'Affiche le message à l'utilisateur
        message = "La destination d'une relation sous-ensemble doit être une classe de parts." _
            & Chr(10) & "Or ce n'est pas le cas de la relation sous-ensemble qui a pour " _
            & "destination la classe " & forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula _
            & "  (page n° " & forme.ContainingPage & ")." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : destination de la relation sous-ensemble"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '3) Vérifie si la relation n'associe pas une classe à elle-même
    If (forme.Connects.Item(1).ToSheet.Data3 Like _
        forme.Connects.Item(2).ToSheet.Data3) Then
        'Affiche le message à l'utilisateur
        message = "La relation sous-ensemble ""isA""" & _
            "  (page n° " & forme.ContainingPage & _
            ")  associe la classe de parts " & _
            forme.Connects.Item(1).ToSheet.Cells("Prop.Nom").Formula & " à elle-même." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : classe de parts associée à elle-même"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '4) Vérifie si la destination est une classe de parts mixtes et l'origine _
    une classe de tampons ou de conteneurs
    If (forme.Connects.Item(2).ToSheet.Data1 Like "BasicOrCompoundPartClass") And _
        ((forme.Connects.Item(1).ToSheet.Data1 Like "Buffer") Or _
        (forme.Connects.Item(1).ToSheet.Data1 Like "Container")) Then
        'Affiche le message à l'utilisateur
        message = "Une classe de parts mixtes  (" & _
            forme.Connects.Item(2).ToSheet.Cells("Prop.Nom").Formula & " à la page n° " _
            & forme.ContainingPage & ")  ne peut pas être la super classe d'une classe de " _
            & "tampons ou de conteneurs  (" _
            & forme.Connects.Item(1).ToSheet.Cells("Prop.Nom").Formula & ")." _
            & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
```

```
            typeErreur = "Erreur : origine de la relation sous-ensemble"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '5) Vérifie si l'origine et la destination sont des classes de parts compatibles
        If Not ((forme.Connects.Item(1).ToSheet.Data1 Like forme.Connects.Item(2).ToSheet.Data1) Or _
            (forme.Connects.Item(2).ToSheet.Data1 Like "BasicOrCompoundPartClass")) Then
            'Affiche le message à l'utilisateur
            message = "La relation sous-ensemble qui a pour origine la classe de parts " _
                & forme.Connects.Item(1).ToSheet.Cells("Prop.Nom").Formula & "  (page n° " _
                & forme.ContainingPage & ")  doit avoir le même type de classe comme destination." _
                & Chr(10) & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : origine de la relation sous-ensemble"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If

    Case Else
        Debug.Print "La classe d'attributs ne rentre pas dans la configuration prévue."

    End Select

End Sub
```

```
'Vérifie les contraintes sur les commentaires
'forme : forme représentant le commentaire à vérifier

Public Sub vérifierContraintesCommentaires(forme As Visio.Shape)

    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer

    'Vérifie si le commentaire est attaché à une classe de parts ou d'états (donc pas isolé)
    If Not (forme.Connects.Count = 1) Then
        'Affiche le message à l'utilisateur
        message = "Un commentaire" & "  (page n° " & forme.ContainingPage & _
                ")  n'est pas attaché à une classe de parts ou d'états." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
        typeErreur = "Erreur : commentaire isolé"
```

```
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If

End Sub
```

```
'Vérifie la syntaxe de la cardinalité d'une relation
'Résolution : passer en revue les cas litigieux possibles
'forme : forme représentant la classe d'attributs à vérifier

Private Function vérifierCardinalité(forme As Visio.Shape)

    'Borne inférieure de la cardinalité
    Dim borneInf As String
    'Borne supérieure de la cardinalité
    Dim borneSup As String
    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer

    'Initialise le type d'erreur et les variables contenant les bornes
    typeErreur = "Erreur : syntaxe de la cardinalité"
    borneInf = forme.Cells("Prop.BorneInférieure").ResultStr(0)
    borneSup = forme.Cells("Prop.BorneSupérieure").ResultStr(0)

    'Vérifie si la borne inférieure est un entier
    If Not IsNumeric(borneInf) Or (borneInf Like "*,*") Then
        'Affiche le message à l'utilisateur
        message = "La borne inférieure d'une relation doit obligatoirement être un entier." _
            & Chr(10) & "Or ce n'est pas le cas de la borne inférieure de la relation " _
            & forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  : " & borneInf & "." & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérifierCardinalité = False
        Exit Function
    End If

    'Vérifie si la borne supérieure est un entier ou le nombre indéfini "N"
    If Not (IsNumeric(borneSup) Or borneSup Like "N") Or (borneSup Like "*,*") Then
        'Affiche le message à l'utilisateur
        message = "La borne supérieure d'une relation doit être un entier ou le nombre indéfini ""N""." _
```

```
            & Chr(10) & "Or ce n'est pas le cas de la borne supérieure de la relation " _
            & forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  : " & borneSup & "." & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérifierCardinalité = False
        Exit Function
End If

'Vérifie que la borne inférieure est un entier positif
If (borneInf < 0) Then
        'Affiche le message à l'utilisateur
        message = "La borne inférieure d'une relation ne peut être négative." _
        & Chr(10) & "Or ce n'est pas le cas de la borne inférieure de la relation " _
            & forme.Cells("Prop.Nom").Formula & "  (page n° " & _
            forme.ContainingPage & ")  : " & borneInf & "." & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérifierCardinalité = False
        Exit Function
End If

'Vérifie que, si la borne supérieure est un entier, celui-ci est positif et non nul
If IsNumeric(borneSup) Then
        If (borneSup < 1) Then
            'Affiche le message à l'utilisateur
            message = "La borne supérieure d'une relation ne peut être ni négative, ni nulle." _
                & Chr(10) & "Or ce n'est pas le cas de la borne supérieure de la relation " _
                & forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  : " & borneSup & "." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérifierCardinalité = False
            Exit Function
        End If
End If

'Vérifie le cas particulier de la borne inférieure de la relation valeur _
qui doit être non nulle
If (forme.Data1 Like "possible_values") Then
        If (borneInf = 0) Then
            'Affiche le message à l'utilisateur
            message = "La borne inférieure de la relation valeur ne peut être nulle." _
                & Chr(10) & "Or ce n'est pas le cas de la borne inférieure de la relation " _
```

```
                & forme.Cells("Prop.Nom").Formula & "  (page n° " & _
                forme.ContainingPage & ")  : " & borneInf & "." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérifierCardinalité = False
            Exit Function
        End If
    End If

    'Vérifie si la borne inférieure n'est pas supérieure à la borne supérieure
    If IsNumeric(borneSup) Then
        If (Cint(borneSup) < Cint(borneInf)) Then
            'Affiche le message à l'utilisateur
            message = "Dans la cardinalité d'une relation, la borne inférieure " & _
                "ne peut être supérieure à la borne supérieure." _
                & Chr(10) & "Or ce n'est pas le cas de cardinalité de la relation " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & forme.ContainingPage _
                & ")  : " & borneInf & ":" & borneSup & "." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérifierCardinalité = False
            Exit Function
        End If
    End If

    'Vérifie si la duplication des valeurs demandée par l'utilisateur est possible
    If (forme.CellExists("Prop.Duplication", 0)) Then
        'Teste si la duplication est demandée et si la borne supérieure n'est pas multiple
        If (forme.Cells("Prop.Duplication").ResultStr(0) Like "VRAI") And _
            (borneSup Like "1") Then
            'Affiche le message à l'utilisateur
            message = "La duplication des valeurs (sac ou séquence) est impossible avec la relation " & _
                forme.Cells("Prop.Nom").Formula & "  (page n° " & forme.ContainingPage _
                & ")" & Chr(10) & _
                "car la borne supérieure de la cardinalité ne permet d'affecter " & _
                "qu'une seule valeur à la classe de dispositifs de valeur." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérifierCardinalité = False
            Exit Function
        End If
    End If
    vérifierCardinalité = True

End Function
```

F.6. Module standard "Module3"

```
'Module nommé "Module3". _
Teste si un modèle graphique de "parts" vérifie les contraintes sur les classes de parts et d'états. _
La correction du modèle graphique est testée lors de sa traduction en langage AlbertII : _
la procédure de ce module est appelée par la macro "traduireEnAlbertII" ("Module1").

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text
```

```
'Vérifie si le nombre et le type de relations par classe de parts et d'états est correct

Public Sub vérifierContraintesPartsEtats()

    'Indice du tableau (dimension1) correspondant au numéro de la classe de parts ou d'états
    Dim num As Integer
    'Compteur de classes de parts et d'états
    Dim i As Integer
    'Compteur de relations (classes d'attributs) et de commentaires
    Dim x As Integer, y As Integer, w As Integer
    'Booléens indiquant l'existence d'une sous-classe de parts de base, composées ou mixtes
    Dim existenceClasseBase As Boolean, existenceClasseComposée As Boolean, _
    existenceClassePile As Boolean, existenceClasseMixte As Boolean
    'Variable contenant la somme des bornes supérieures des relations dispositifs géométriques
    Dim somme As Integer
    'Booléen indiquant si la somme évoquée ci-dessus est supérieure ou égale à deux
    Dim sommeSuffisante As Boolean
    'Booléen indiquant si une classe de parts est le composant d'une classe de parts composées _
    définie par une classe de dispositifs géométriques.
    Dim composée As Boolean
    'Variable contenant la borne supérieure ou inférieure d'une classe d'attributs composants
    Dim borne As String
    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer

    'Pour toutes les classes de parts et d'états
    For i = 1 To tabParts(0, 0, 0)

        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, i)

        'Renvoie le nom de la méta-classe dont la classe de parts ou d'états est l'instance _
        ou le nom de la classe spéciale dont la classe de parts ou d'états est le sous-ensemble
```

```
Select Case tabParts(num, 0, 1)

'Classe de parts
Case "BasicPartClass", "CompoundPartClass", "BasicOrCompoundPartClass", _
    "PileOfParts", "Container", "Buffer"
    '1) Vérifie si la classe est le sous-ensemble d'une seule autre classe. _
    La classe doit donc être l'origine de maximum une relation sous-ensemble.
    'Teste s'il existe plusieurs relations.
    If (tabParts(num, 0, 0) > 1) Then
        'Toutes les relations sous-ensembles sont au début de la dimension2 du tableau
        If (tabParts(num, 2, 0) Like "isAOrigine") Then
            'Affiche le message à l'utilisateur
            message = "Une classe de parts ne peut être le sous-ensemble que " _
                & "d'une seule classe de parts." _
                & Chr(10) & "Or ce n'est pas le cas de la classe de parts """ & _
                tabParts(num, 0, 2) & """." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : héritage multiple"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '2) Vérifie qu'une part ne possède pas des identités multiples. _
    La classe de parts doit donc être l'origine d'au moins une relation identité _
    sauf si elle est un sous-ensemble (car une sous-classe hérite _
    des classes d'identités de ses super classes).
    'Teste si la classe possède une identité ou est un sous-ensemble
    If Not (posséderRelation(num, "possible_identities") Xor _
        posséderRelation(num, "isAOrigine")) Then
        'Teste si la classe possède une identité
        If posséderRelation(num, "possible_identities") Then
            'Affiche le message à l'utilisateur
            message = "Une sous-classe de parts ne peut être associée à une classe " _
                & "d'identités" & Chr(10) & "car elle hérite de la classe " _
                & "d'identités de sa super classe racine." & Chr(10) & _
                "Or la classe de parts sous-ensemble """ & tabParts(num, 0, 2) & _
                """ possède une classe d'identités." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : sous-classe de parts multi-identité"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        Else
            'Affiche le message à l'utilisateur
            message = "Chaque part doit posséder une identité." _
```

```
                & Chr(10) & "Or la classe de parts """ & tabParts(num, 0, 2) & _
                """ n'est associée à aucune classe d'identités." & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : parts sans identité"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '3) Vérifie qu'une part ne possède pas des positions multiples. _
    La classe de parts ne peut donc pas être l'origine d'une relation position _
    si elle est un sous-ensemble.
    'Teste si la classe est un sous-ensemble et possède une classe de positions
    If (posséderRelation(num, "isAOrigine") And _
        posséderRelation(num, "possible_positions")) Then
        'Affiche le message à l'utilisateur
        message = "Une sous-classe de parts ne peut être associée à une classe de positions" _
            & Chr(10) & "car elle doit hériter la classe de positions de sa super classe." _
            & Chr(10) & "Or la classe de parts sous-ensemble """ & tabParts(num, 0, 2) & _
            """ possède une classe de positions." & Chr(10) & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : parts multi-positions"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    End If
    '4) Vérifie qu'une part composée ou une pile peut être constituée de plusieurs composants.
    'Teste s'il s'agit d'une classe de parts feuille ou non sous-ensemble
    If ((tabParts(num, 0, 1) Like "CompoundPartClass") Or _
        (tabParts(num, 0, 1) Like "PileOfParts")) And _
        Not (posséderRelation(num, "isADestination")) Then
        'Teste si la contrainte est vérifiée (voir "Module5")
        vérifierComposants num, "possible_components"
        If Not (vérification) Then
            'Affiche le message à l'utilisateur
            borne = Switch(tabParts(num, 0, 1) Like "CompoundPartClass", "part composée", _
                tabParts(num, 0, 1) Like "PileOfParts", "pile")
            typeErreur = Switch(tabParts(num, 0, 1) Like "CompoundPartClass", _
                "parts composées", tabParts(num, 0, 1) Like "PileOfParts", "piles")
            message = "Une " & borne & " doit être constituée de plusieurs composants." _
                & Chr(10) & "A cet effet, la classe de " & typeErreur & " et/ou ses super " _
                & "classes éventuelles doivent posséder une ou plusieurs classes d'attributs " _
                & "composants dont la somme des bornes supérieures des cardinalités " _
                & "est égale ou supérieure à deux." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de " & typeErreur & " """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
```

```
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : " & borne & " sans composants"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '5) Vérifie qu'un conteneur peut contenir plusieurs contenus.
    'Teste s'il s'agit d'une classe de conteneurs feuille ou non sous-ensemble
    If (tabParts(num, 0, 1) Like "Container") And _
        Not (posséderRelation(num, "isADestination")) Then
        'Teste si la contrainte est vérifiée (voir "Module5")
        vérifierComposants num, "contain"
        If Not (vérification) Then
            'Affiche le message à l'utilisateur
            message = "Un conteneur doit contenir plusieurs contenus." & Chr(10) _
                & "A cet effet, la classe de conteneurs et/ou ses super classes " _
                & "éventuelles doivent posséder une ou plusieurs classes d'attributs " _
                & "contenants dont la somme des bornes supérieures des cardinalités " _
                & "est égale ou supérieure à deux." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de conteneurs """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : conteneur sans contenus"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '6) Vérifie la contrainte sous-ensemble spécifique à la classe de parts mixtes
    'Teste s'il s'agit d'une classe de parts mixtes
    If (tabParts(num, 0, 1) Like "BasicOrCompoundPartClass") Then
        'Une classe de parts mixtes doit être une super classe et doit posséder ...
        If posséderRelation(num, "isADestination") Then
            'Initialise les variables booléennes
            existenceClasseBase = False
            existenceClasseComposée = False
            existenceClassePile = False
            existenceClasseMixte = False
            'Pour toutes les classes d'attributs
            For x = 1 To tabParts(num, 0, 0)
                If (tabParts(num, x, 0) Like "isADestination") Then
                    '... soit une sous-classe directe de parts mixtes.
                    If (tabParts(tabParts(num, x, 4), 0, 1) Like _
                        "BasicOrCompoundPartClass") Then
                        existenceClasseMixte = True
```

```
                    End If
                    '... soit deux sous-classes directes de types différents : _
                    part de base, part composée ou pile.
                    If (tabParts(tabParts(num, x, 4), 0, 1) Like "BasicPartClass") Then
                        existenceClasseBase = True
                    End If
                    If (tabParts(tabParts(num, x, 4), 0, 1) Like "CompoundPartClass") Then
                        existenceClasseComposée = True
                    End If
                    If (tabParts(tabParts(num, x, 4), 0, 1) Like "PileOfParts") Then
                        existenceClassePile = True
                    End If
                End If
            Next x
        Else
            message = "Une classe mixte doit obligatoirement être une super classe." _
                & Chr(10) & "Or ce n'est pas le cas de la classe mixte """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : classe de parts mixtes non super classe"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        'Teste si aucune des deux options n'est vérifiée
        If Not (existenceClasseMixte) And _
            Not ((existenceClasseBase And existenceClasseComposée) Or _
            (existenceClasseBase And existenceClassePile) Or _
            (existenceClasseComposée And existenceClassePile)) Then
            message = "Une classe mixte doit obligatoirement posséder " _
                & "soit une sous-classe de parts mixtes," & Chr(10) & _
                "soit deux sous-classes directes de types différents " _
                & "(part de base, part composée ou pile)." _
                & Chr(10) & "Or ce n'est pas le cas de la classe mixte """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : emploi abusif de la classe de parts mixtes"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '7) Vérifie qu'un tampon peut contenir des contenus. _
    La classe doit donc être l'origine d'au moins une relation contenu.
    'Teste s'il ne s'agit pas d'une super classe
    If (tabParts(num, 0, 1) Like "Buffer") And _
```

```
        Not (posséderRelation(num, "isADestination")) Then
        'Teste si la classe possède une classe d'attributs contenus
        If (compterRelation(num, "possible_contents") = 0) Then
            'Affiche le message à l'utilisateur
            message = "Une classe de tampons doit posséder au moins une classe " _
                & "d'attributs contenus." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de tampons """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : tampons sans contenus possibles"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '8) Vérifie qu'une pile ou qu'un conteneur possède au moins un dispositif de contiguïté. _
    La classe doit donc être l'origine d'au moins une relation contiguïté.
    'Teste s'il ne s'agit pas d'une super classe
    If ((tabParts(num, 0, 1) Like "PileOfParts") Or _
        (tabParts(num, 0, 1) Like "Container")) And _
        Not (posséderRelation(num, "isADestination")) Then
        'Teste si la classe possède une classe d'attributs contiguïté
        If (compterRelation(num, "contiguity") = 0) Then
            'Affiche le message à l'utilisateur
            borne = Switch(tabParts(num, 0, 1) Like "Container", "conteneur", _
                tabParts(num, 0, 1) Like "PileOfParts", "pile")
            message = "Une classe de " & borne & "s doit posséder au moins " _
                & "une classe d'attributs contiguïté." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de " & borne & "s """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : " & borne & " sans dispositif de contiguïté"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
    '9) Vérifie qu'une pile ou qu'un conteneur possède maximum un dispositif d'enclos. _
    La classe doit donc être l'origine de maximum une relation enclos.
    'Teste s'il ne s'agit pas d'une super classe
    If ((tabParts(num, 0, 1) Like "PileOfParts") Or _
        (tabParts(num, 0, 1) Like "Container")) And _
        Not (posséderRelation(num, "isADestination")) Then
        'Comptabilise le nombre de classes d'attributs enclos
        If Not (compterRelation(num, "enclosure") < 2) Then
            'Affiche le message à l'utilisateur
```

```
                borne = Switch(tabParts(num, 0, 1) Like "Container", "conteneur", _
                    tabParts(num, 0, 1) Like "PileOfParts", "pile")
                message = "Une classe de " & borne & "s ne peut posséder qu'une seule " _
                    & "classe d'attributs enclos." & Chr(10) & "Or la classe de " _
                    & borne & "s """ & tabParts(num, 0, 2) & """ en possède " _
                    & compterRelation(num, "enclosure") & "." & Chr(10) & Chr(10) & _
                    "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : " & borne & " sans dispositif d'enclos unique"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Sub
            End If
        End If
        '10) Vérifie que les piles d'une même hiérarchie sont du même type. _
        Les types d'une super classe de piles et de ses sous-classes doivent être identiques.
        'Teste s'il s'agit d'une super classe de piles
        If (tabParts(num, 0, 1) Like "PileOfParts") And _
            (posséderRelation(num, "isADestination")) Then
            'Pour toutes les classes d'attributs de la super classe de piles
            For x = 1 To tabParts(num, 0, 0)
                'Teste s'il s'agit d'une classe d'attributs sous-classes
                If (tabParts(num, x, 0) Like "isADestination") Then
                    'Teste si les types des deux classes sont différents
                    If Not (tabParts(num, 0, 3) Like tabParts(tabParts(num, x, 4), 0, 3)) Then
                        'Affiche le message à l'utilisateur
                        message = "La super classe de piles """ & tabParts(num, 0, 2) _
                            & """ et sa sous-classe """ & tabParts(tabParts(num, x, 4), 0, 2) _
                            & """ doivent être du même type : " & tabParts(num, 0, 3) & " ou " _
                            & tabParts(tabParts(num, x, 4), 0, 3) & "." & Chr(10) & Chr(10) & _
                            "Veuillez corriger l'erreur avant de recommencer la traduction." _
                            & Chr(10) & Chr(10)
                        typeErreur = "Erreur : piles de types différents"
                        z = MsgBox(message, 0, typeErreur)
                        'Force l'arrêt de la macro "traduireEnAlbertII"
                        vérification = False
                        Exit Sub
                    End If
                End If
            Next x
        End If

    'Classe d'identités
    Case "PartIdentityClass"
        'Vérifie si la classe est associée à une classe de parts. _
        La classe doit donc être la destination d'une relation identité.
        If Not (vérifierExistenceRelation(num, "identités")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
```

```
            Exit Sub
        End If

    'Classe de positions
    Case "PartPositionClass"
        'Vérifie si la classe est associée à une classe de parts. _
        La classe doit donc être la destination d'une relation position.
        If Not (vérifierExistenceRelation(num, "positions")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If

    'Classe de dispositifs physiques
    Case "PartPhysicalFeatureClass"
        'Vérifie si la classe est associée à une classe de parts. _
        La classe doit donc être la destination d'une relation dispositif physique.
        If Not (vérifierExistenceRelation(num, "dispositifs physiques")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If

    'Classe de dispositifs géométriques
    Case "PartGeometricalFeatureClass"
        '1) Vérifie si la classe est associée à une classe de parts composées. _
        La classe doit donc être la destination d'une relation géométrie.
        If Not (vérifierExistenceRelation(num, "dispositifs géométriques")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la classe est associée à une classe de parts (composantes). _
        La classe doit donc être l'origine d'une relation dispositif géométrique.
        If Not (posséderRelation(num, "possible_geometrical_features")) Then
            message = "Une classe de dispositifs géométriques doit être associée à " _
                & "au moins une classe de parts composantes." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de dispositifs géométriques """ & _
                tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : classe de dispositifs géométriques isolée"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie que les classes de parts composantes associées à la classe _
        de dispositifs géométriques sont associées à la classe de parts composées _
        qui est définie par cette classe de dispositifs géométriques.
```

```
'Pour toutes les classes d'attributs de la classe de dispositifs géométriques
For x = 1 To tabParts(num, 0, 0)
    'Teste s'il s'agit d'une classe d'attributs dispositifs géométriques
    If (tabParts(num, x, 0) Like "possible_geometrical_features") Then
        'Teste si la classe de parts n'est pas un composant
        If Not (vérifierDispositifs(tabDispositifs(num, 0), tabParts(num, x, 4))) Then
            'Affiche le message à l'utilisateur
            message = "Toutes les classes de parts composantes associées à une classe " _
                & "de dispositifs géométriques doivent être liées à la classe de parts " _
                & "composées qui est définie par cette classe de dispositifs géométriques." _
                & Chr(10) & "Or ce n'est pas le cas de la classe de parts composantes """ _
                & tabParts(tabParts(num, x, 4), 0, 2) & """ qui définit la classe de dispositifs " _
                & "géométriques """ & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : dispositif géométrique assemblant des parts étrangères"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
    End If
Next x
'4) Vérifie que les bornes de la cardinalité de chaque classe d'attributs _
dispositifs géométriques sont inférieures ou égales aux bornes de la cardinalité _
de la classe d'attributs composants dont la destination est la même que cette classe.
'Pour toutes les classes d'attributs de la classe de dispositifs géométriques
For x = 1 To tabParts(num, 0, 0)
    'Teste s'il s'agit d'une classe d'attributs dispositifs géométriques
    If (tabParts(num, x, 0) Like "possible_geometrical_features") Then
        'Pour chaque borne de la classe d'attributs dispositifs géométriques
        For y = 2 To 3
            'Renvoie la borne de la cardinalité de la classe d'attributs composants
            borne = chercherBorne(tabParts(num, x, 4), tabDispositifs(num, 0), _
                Switch(y Like 2, "inférieure", y Like 3, "supérieure"))
            'Initialise le booléen indiquant si la contrainte est vérifiée
            composée = True
            'Teste si la borne de la classe d'attributs composants est un entier
            If IsNumeric(borne) Then
                'Teste si la borne de la classe d'attributs dispositifs est indéfinie
                If Not (IsNumeric(tabParts(num, x, y))) Then
                    composée = False
                'La borne de la classe d'attributs dispositifs géométriques est un entier
                Else
                    'Teste si la borne de la classe d'attributs dispositifs est supérieure
                    If (Cint(tabParts(num, x, y)) > Cint(borne)) Then
                        composée = False
                    End If
                End If
            End If
```

```
                    'Teste si la contrainte n'est pas vérifiée
                    If Not (composée) Then
                        'Affiche le message à l'utilisateur
                        message = "La borne " & Switch(y Like 2, "inférieure", y Like 3, _
                            "supérieure") & " (" & tabParts(num, x, y) & ") de la cardinalité " _
                            & "de la classe d'attributs dispositifs géométriques """ _
                            & tabParts(num, x, 1) & """ ne peut pas être supérieure à la borne " & _
                            Switch(y Like 2, "inférieure", y Like 3, "supérieure") _
                            & " (" & borne & ") de la cardinalité de la classe " & _
                            "d'attributs composants dont la destination est """ _
                            & tabParts(tabParts(num, x, 4), 0, 2) _
                            & """ ou une de ses super classes." & Chr(10) & Chr(10) & _
                            "Veuillez corriger l'erreur avant de recommencer la traduction." _
                            & Chr(10) & Chr(10)
                        typeErreur = "Erreur : dispositif géométrique assemblant trop de parts"
                        z = MsgBox(message, 0, typeErreur)
                        'Force l'arrêt de la macro "traduireEnAlbertII"
                        vérification = False
                        Exit Sub
                    End If
                Next y
            End If
        Next x
        '5) Vérifie qu'un dispositif géométrique peut assembler plusieurs parts. _
        Donc la somme des bornes supérieures des cardinalités des relations dispositifs _
        géométriques d'une classe de dispositifs géométriques doit être supérieure ou égale à deux.
        'Initialise les variables
        somme = 0
        sommeSuffisante = False
        'Pour toutes les classes d'attributs de la classe de dispositifs géométriques
        For x = 1 To tabParts(num, 0, 0)
            'Teste s'il s'agit d'une relation dispositifs géométriques
            If (tabParts(num, x, 0) Like "possible_geometrical_features") Then
                'Teste si la borne supérieure est un nombre indéfini
                If (tabParts(num, x, 3) Like "N") Then
                    sommeSuffisante = True
                Else
                    'Sinon comptabilise les bornes
                    somme = somme + tabParts(num, x, 3)
                End If
            End If
        Next x
        'Teste si la somme des bornes supérieures n'est pas supérieure ou égale à deux
        If Not (sommeSuffisante) And Not (somme >= 2) Then
            'Affiche le message à l'utilisateur
            message = "Un dispositif géométrique doit assembler plusieurs parts composantes." _
                & Chr(10) & "A cet effet, la classe de dispositifs géométriques " _
                & "doit posséder une ou plusieurs classes de dispositifs géométriques " _
                & "dont la somme des bornes supérieures des cardinalités " _
                & "est égale ou supérieure à deux." & Chr(10) & _
```

```
                "Or ce n'est pas le cas de la classe de dispositifs géométriques """ _
                & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : dispositif géométrique assemblant une seule part"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If

'Classe de dispositifs de fixation
Case "PartFixingFeatureClass"
        '1) Vérifie si la classe est associée à une classe de dispositifs géométriques. _
        La classe doit donc être l'origine d'une relation dispositif de fixation.
        If Not (posséderRelation(num, "possible_fixing_features")) Then
            message = "Une classe de dispositifs de fixation doit être associée à " _
                & "au moins une classe de dispositifs géométriques." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de dispositifs de fixation """ & _
                tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : classe de dispositifs de fixation isolée"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie que toutes les classes de dispositifs géométriques qui définissent _
        la classe de dispositifs de fixation définissent la même classe de parts composées. _
        Vérifie également que les les bornes de la cardinalité de la classe d'attributs _
        dispositifs de fixation ne sont pas plus élevées que les bornes de la cardinalité _
        de la classe d'attributs géométries dont la destination est la même que cette classe.
        'Initialise la variable contenant le numéro de la classe de parts composées _
        définie par la première classe de dispositifs géométriques
        somme = 0
        'Pour toutes les classes d'attributs de la classe de dispositifs de fixation
        For x = 1 To tabParts(num, 0, 0)
            'Teste s'il s'agit d'une classe d'attributs dispositifs de fixation
            If (tabParts(num, x, 0) Like "possible_fixing_features") Then
                '2.1) Vérifie la première partie de la contrainte
                'Teste s'il s'agit de la première classe de dispositifs géométriques
                If (somme = 0) Then
                    'Enregistre le numéro de la classe de parts composées définie
                    somme = tabDispositifs(tabParts(num, x, 4), 0)
                Else
                    'Vérifie si les classes de dispositifs géométriques ne définissent _
                    pas la même classe de parts composées
                    If Not (tabDispositifs(tabParts(num, x, 4), 0) Like somme) Then
                        'Affiche le message à l'utilisateur
```

```
                message = "Toutes les classes de dispositifs géométriques qui " _
                    & "définissent une classe de dispositifs de fixation doivent " _
                    & "définir la même classe de parts composées." & Chr(10) _
                    & "Or les classes de dispositifs géométriques de la classe de " _
                    & "fixation """ & tabParts(num, 0, 2) & """ définissent les classes " _
                    & "de parts composées """ & tabParts(somme, 0, 2) & """ et """ _
                    & tabParts(tabDispositifs(tabParts(num, x, 4), 0), 0, 2) & """." _
                    & Chr(10) & Chr(10) _
                    & "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : dispositif de fixation défini par des dispositifs " _
                    & "géométriques étrangers"
                z = MsgBox(message, 0, typeErreur)
                'Force l'arrêt de la macro "traduireEnAlbertII"
                vérification = False
                Exit Sub
            End If
        End If
        '2.2) Vérifie la seconde partie de la contrainte
        'Pour chaque borne de la classe d'attributs dispositifs de fixation
        For y = 2 To 3
            'Initialise le booléen indiquant si la contrainte est vérifiée
            composée = True
            'Teste si la borne de la classe d'attributs géométries est un entier
            If IsNumeric(tabDispositifs(tabParts(num, x, 4), y - 1)) Then
                'Teste si la borne de la classe d'attributs dispositifs est indéfinie
                If Not (IsNumeric(tabParts(num, x, y))) Then
                    composée = False
                'La borne de la classe d'attributs dispositifs géométriques est un entier
                Else
                    'Teste si la borne de la classe d'attributs dispositifs est supérieure
                    If (Cint(tabParts(num, x, y)) > _
                        Cint(tabDispositifs(tabParts(num, x, 4), y - 1))) Then
                        composée = False
                    End If
                End If
            End If
            'Teste si la contrainte n'est pas vérifiée
            If Not (composée) Then
                'Affiche le message à l'utilisateur
                message = "La borne " & Switch(y Like 2, "inférieure", y Like 3, _
                    "supérieure") & " (" & tabParts(num, x, y) & ") de la cardinalité " _
                    & "de la classe d'attributs dispositifs de fixation """ _
                    & tabParts(num, x, 1) & """ ne peut pas être supérieure à la borne " & _
                    Switch(y Like 2, "inférieure", y Like 3, "supérieure") & " (" _
                    & tabDispositifs(tabParts(num, x, 4), y - 1) & ") de la cardinalité " _
                    & "de la classe d'attributs géométries dont la destination est """ _
                    & tabParts(tabParts(num, x, 4), 0, 2) & """." & Chr(10) & Chr(10) & _
                    "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
```

```
                        typeErreur = "Erreur : dispositif de fixation définissant trop de " _
                            & "dispositifs géométriques"
                        z = MsgBox(message, 0, typeErreur)
                        'Force l'arrêt de la macro "traduireEnAlbertII"
                        vérification = False
                        Exit Sub
                    End If
                Next y
            End If
        Next x

'Classe de dispositifs de contiguïté
Case "PartContiguityFeature"
        '1) Vérifie qu'un dispositif de contiguïté assemble toujours deux parts. _
        Donc la somme des bornes inférieures/supérieures des cardinalités des relations _
        dispositifs de contiguïté d'une classe de dispositifs de contiguïté doit égaler deux.
        'Initialise la variable contenant la somme des bornes supérieures
        somme = 0
        'Pour toutes les classes d'attributs de la classe de dispositifs de contiguïté
        For x = 1 To tabParts(num, 0, 0)
            'Teste s'il s'agit d'une classe d'attributs dispositifs de contiguïté
            If (tabParts(num, x, 0) Like "contiguity_feature") Then
                'Comptabilise les bornes
                somme = somme + tabParts(num, x, 3)
            End If
        Next x
        'Teste si la somme des bornes supérieures (donc aussi inférieures) est égale à deux
        If (somme <> 2) Then
            'Affiche le message à l'utilisateur
            message = "Un dispositif de contiguïté doit toujours rendre deux parts contiguës." _
                & Chr(10) & "A cet effet, la classe de dispositifs de contiguïté " _
                & "doit posséder une ou deux classes de dispositifs de contiguïté " _
                & "dont la somme des bornes inférieures/supérieures des cardinalités " _
                & "est égale à deux." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de dispositifs de contiguïté """ _
                & tabParts(num, 0, 2) & """ dont la somme des bornes vaut : " _
                & somme & "." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : dispositif de contiguïté rendant moins ou plus de deux " _
                & "parts contiguës"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la classe est associée à une classe de piles ou de conteneurs. _
        La classe doit donc être la destination d'une relation contiguïté.
        If Not (vérifierExistenceRelation(num, "dispositifs de contiguïté")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
```

```
            vérification = False
            Exit Sub
        End If
        '3) Vérifie que les classes de parts composantes/contenues associées à la classe _
        de dispositifs de contiguïté sont associées à la classe de piles/conteneurs _
        qui est définie par cette classe de dispositifs de contiguïté.
        'Pour toutes les classes d'attributs de la classe de dispositifs de contiguïté
        For x = 1 To tabParts(num, 0, 0)
            'Teste s'il s'agit d'une classe d'attributs dispositifs de contiguïté
            If (tabParts(num, x, 0) Like "contiguity_feature") Then
                'Teste si la classe de parts n'est pas un composant/contenu
                If Not (vérifierDispositifs(tabDispositifs(num, 0), tabParts(num, x, 4))) Then
                    borne = Switch(tabParts(tabDispositifs(num, 0), 0, 1) Like "PileOfParts", "piles", _
                        tabParts(tabDispositifs(num, 0), 0, 1) Like "Container", "conteneurs")
                    typeErreur = Switch(tabParts(tabDispositifs(num, 0), 0, 1) Like "PileOfParts", _
                        "composantes", tabParts(tabDispositifs(num, 0), 0, 1) Like "Container", _
                        "contenues")
                    'Affiche le message à l'utilisateur
                    message = "Toutes les classes de parts " & typeErreur & " associées à une " _
                        & "classe de dispositifs de contiguïté doivent être liées à la classe de " _
                        & borne & " qui est définie par cette classe de dispositifs " _
                        & "de contiguïté." & Chr(10) & "Or ce n'est pas le cas de la classe " _
                        & "de parts " & typeErreur & " """ & tabParts(tabParts(num, x, 4), 0, 2) _
                        & """ qui définit la classe de dispositifs de contiguïté """ _
                        & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                        "Veuillez corriger l'erreur avant de recommencer la traduction." _
                        & Chr(10) & Chr(10)
                    typeErreur = "Erreur : dispositif de contiguïté rendant des parts étrangères contiguës"
                    z = MsgBox(message, 0, typeErreur)
                    'Force l'arrêt de la macro "traduireEnAlbertII"
                    vérification = False
                    Exit Sub
                End If
            End If
        Next x

    'Classe de dispositifs d'enclos
    Case "PartEnclosureFeature"
        '1) Vérifie si la classe est associée à une classe de parts composantes/contenues. _
        La classe doit donc être l'origine d'une relation dispositif d'enclos.
        If Not (posséderRelation(num, "enclosure_feature")) Then
            message = "Une classe de dispositifs d'enclos doit être associée à " _
                & "au moins une classe de parts composantes ou contenues." & Chr(10) & _
                "Or ce n'est pas le cas de la classe de dispositifs d'enclos """ & _
                tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : dispositif d'enclos qui n'enclot rien"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
```

```
            vérification = False
            Exit Sub
        End If
        '2) Vérifie si la classe est associée à une classe de piles ou de conteneurs. _
        La classe doit donc être la destination d'une relation enclos.
        If Not (vérifierExistenceRelation(num, "dispositifs d'enclos")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        End If
        '3) Vérifie que les classes de parts composantes/contenues associées à la classe _
        de dispositifs d'enclos sont associées à la classe de piles/conteneurs _
        qui est définie par cette classe de dispositifs d'enclos.
        'Pour toutes les classes d'attributs de la classe de dispositifs d'enclos
        For x = 1 To tabParts(num, 0, 0)
            'Teste s'il s'agit d'une classe d'attributs dispositifs d'enclos
            If (tabParts(num, x, 0) Like "enclosure_feature") Then
                'Teste si la classe de parts n'est pas un composant/contenu
                If Not (vérifierDispositifs(tabDispositifs(num, 0), tabParts(num, x, 4))) Then
                    borne = Switch(tabParts(tabDispositifs(num, 0), 0, 1) Like "PileOfParts", "piles", _
                        tabParts(tabDispositifs(num, 0), 0, 1) Like "Container", "conteneurs")
                    typeErreur = Switch(tabParts(tabDispositifs(num, 0), 0, 1) Like "PileOfParts", _
                        "composantes", tabParts(tabDispositifs(num, 0), 0, 1) Like "Container", _
                        "contenues")
                    'Affiche le message à l'utilisateur
                    message = "Toutes les classes de parts " & typeErreur & " associées à une " _
                        & "classe de dispositifs d'enclos doivent être liées à la classe de " _
                        & borne & " qui est définie par cette classe de dispositifs " _
                        & "d'enclos." & Chr(10) & "Or ce n'est pas le cas de la classe " _
                        & "de parts " & typeErreur & " """ & tabParts(tabParts(num, x, 4), 0, 2) _
                        & """ qui définit la classe de dispositifs d'enclos """ _
                        & tabParts(num, 0, 2) & """." & Chr(10) & Chr(10) & _
                        "Veuillez corriger l'erreur avant de recommencer la traduction." _
                        & Chr(10) & Chr(10)
                    typeErreur = "Erreur : dispositif d'enclos enclosant des parts étrangères"
                    z = MsgBox(message, 0, typeErreur)
                    'Force l'arrêt de la macro "traduireEnAlbertII"
                    vérification = False
                    Exit Sub
                End If
            End If
        Next x

    'Classe de dispositifs de valeur
    Case "PartFeatureValueClass"
        'Vérifie si la classe est associée à une classe de dispositifs. _
        La classe doit donc être la destination d'une relation valeur.
        If Not (vérifierExistenceRelation(num, "dispositifs de valeur")) Then
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
```

```
                Exit Sub
            End If

        Case Else
            Debug.Print "La classe de parts ou d'états ne rentre pas dans la configuration prévue."

        End Select

    Next i

End Sub
```

```
'Envoie un message à l'utilisateur si une classe d'états est isolée (sans relation)
'num : numéro associé à la classe d'états
'nom : nom de la méta-classe/classe spéciale dont la classe d'états est l'instance/le sous-ensemble

Private Function vérifierExistenceRelation(num As Integer, nom As String)

    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer

    If (tabRelations(num) Like "") Then
        'Affiche le message à l'utilisateur
        message = "La classe d'états " & nom & " """ & tabParts(num, 0, 2) _
            & """ ne possède pas de relations " & Switch(nom Like "dispositifs de contiguïté", _
            "contiguïtés dont elle est la destination", nom Like "dispositifs d'enclos", _
            "enclos dont elle est la destination", nom Like "dispositifs géométriques", _
            "géométries", True, nom) & "." & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : classe d'états " & nom & " isolée"
        z = MsgBox(message, 0, typeErreur)
        vérifierExistenceRelation = False
    Else
        vérifierExistenceRelation = True
    End If

End Function
```

```
'Renvoie la borne d'une classe d'attributs composants dont la destination est "num"
'num : numéro de la classe de parts composantes
'composé : numéro de la (super) classe de parts composées de la classe de parts composantes
'typeBorne : indique si la borne supérieure ou inférieure est recherchée
```

```
Private Function chercherBorne(num As String, composé As String, typeBorne As String)

    'Variable contenant la borne de la classe d'attributs composants
    Dim borne As String
    'Compteur de classes d'attributs
    Dim i As Integer

    'Initialise la borne
    borne = ""
    '1) Cherche la classe d'attributs au sein de la classe de parts composées
    'Pour toutes les classes d'attributs de la classe de parts composées
    For i = 1 To tabParts(composé, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs composants dont la destination est _
        la classe d'attributs composantes
        If (tabParts(composé, i, 4) Like num) Then
            'Teste le type de borne recherché
            If (typeBorne Like "supérieure") Then
                chercherBorne = tabParts(composé, i, 3)
            Else
                chercherBorne = tabParts(composé, i, 2)
            End If
            Exit Function
        End If
    Next i
    '2) Relance la fonction avec la super classe de la classe de parts composantes
    'Teste si la classe de parts composantes possède une super classe
    If (tabParts(num, 1, 0) Like "isAOrigine") Then
        borne = chercherBorne(tabParts(num, 1, 4), composé, typeBorne)
    End If
    'Teste si la borne a été trouvée
    If Not (borne Like "") Then
        chercherBorne = borne
        Exit Function
    End If
    '3) Relance la fonction avec la super classe de la classe de parts composées
    'Teste si la classe de parts composantes possède une super classe
    If (tabParts(composé, 1, 0) Like "isAOrigine") Then
        borne = chercherBorne(num, tabParts(composé, 1, 4), typeBorne)
    End If
    'Teste si la borne a été trouvée
    If Not (borne Like "") Then
        chercherBorne = borne
        Exit Function
    End If
    '4) Relance la fonction avec toutes les sous-classes de la classe de parts composées
    chercherBorne = chercherBorneMax(num, composé, typeBorne)

End Function
```

```
'Renvoie la borne maximale d'une classe d'attributs composants dont la destination est "num"
'num : numéro de la classe de parts composantes
'composé : numéro de la (super) classe de parts composées de la classe de parts composantes
'typeBorne : indique si la borne supérieure ou inférieure est recherchée

Private Function chercherBorneMax(num As String, composé As String, typeBorne As String)

    'Variable contenant la borne maximale de la classe d'attributs composants
    Dim borneMax As String
    'Variable contenant la borne de la classe d'attributs composants
    Dim borne As String
    'Compteur de classes d'attributs
    Dim i As Integer

    'Initialise la borne maximale
    borneMax = ""
    '1) Cherche la classe d'attributs au sein de la classe de parts composées
    'Pour toutes les classes d'attributs de la classe de parts composées
    For i = 1 To tabParts(composé, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs composants dont la destination est _
        la classe d'attributs composantes
        If (tabParts(composé, i, 4) Like num) Then
            'Teste le type de borne recherché
            If (typeBorne Like "supérieure") Then
                chercherBorneMax = tabParts(composé, i, 3)
            Else
                chercherBorneMax = tabParts(composé, i, 2)
            End If
            Exit Function
        End If
    Next i
    '2) Relance la fonction avec toutes les sous-classes de la classe de parts composées
    'Initialise la borne maximale
    borneMax = 0
    'Pour toutes les classes d'attributs de la classe de parts composées
    For i = 1 To tabParts(composé, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs sous-classes destination
        If (tabParts(composé, i, 0) Like "isADestination") Then
            'Lance la recherche de la borne maximale
            borne = chercherBorneMax(num, tabParts(composé, i, 4), typeBorne)
            'Teste si la borne de la (sous-)classe est un chiffre indéfini
            If (borneMax Like "n") Or (borne Like "n") Then
                'Renvoie la borne
                chercherBorneMax = "n"
                Exit Function
            Else
            'Les deux bornes sont des entiers
                'Teste si la borne de la relation composant de la sous-classe est supérieure
                If (borne > borneMax) Then
                    borneMax = borne
```

```
                End If
            End If
        End If
    Next i
    chercherBorneMax = borne

End Function
```

```
'Renvoie un booléen indiquant si la classe de parts "destination" est un composant ou
'un contenu de la classe de piles, de parts composées ou de conteneurs "origine"
'origine : numéro de la classe de piles, de parts composées ou de conteneurs
'destination : numéro de la classe de parts qui fait l'objet de la vérification

Private Function vérifierDispositifs(origine As String, destination As String)

    'Compteur de parts composées au sens large d'une classe de parts composantes/contenues
    Dim i As Integer
    'Booléen indiquant si la classe de parts est un composant ou un contenu de la classe _
    de parts composées au sens large
    Dim composée As Boolean

    'Initialise le booléen indiquant si la classe de parts composantes/contenues _
    est un composant/contenu de la classe de parts composées au sens large.
    composée = False
    'Pour toutes les classes de parts composées au sens large de la classe de parts
    For i = 1 To tabComposants(destination, 0)
        'Teste si la classe de parts composantes est un composant
        If (tabComposants(destination, i) Like origine) Then
            composée = True
        End If
    Next i
    'Teste si la classe de parts n'est pas un composant/contenu
    If Not (composée) Then
        'Relance la recherche pour toutes les sous-classes de la classe de parts composées
        'Pour toutes les classes d'attributs de la classe de parts composées au sens large
        For i = 1 To tabParts(origine, 0, 0)
            'Teste s'il s'agit d'une classe d'attributs sous-classes destination
            If (tabParts(origine, i, 0) Like "isADestination") Then
                'Teste si la classe de parts n'est pas un composant/contenu de la sous-classe
                If Not (vérifierDispositifs(tabParts(origine, i, 4), destination)) Then
                    vérifierDispositifs = False
                    Exit Function
                Else
                    composée = True
                End If
            End If
        Next i
    End If
    vérifierDispositifs = composée

End Function
```

```
'Renvoie le nombre de relations "relation" que la classe de parts ou d'états "num" possède,
'soit directement soit par héritage.
'num : numéro de la classe de parts ou d'états dont les relations sont comptabilisées.
'relation : type de relation (classe d'attributs).

Public Function compterRelation(num As Integer, relation As String)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Nombre de relations "relation" que la classe de parts ou d'états possède
    Dim nombreRelations As Integer
    'Numéro de la super classe de la classe de parts ou d'états testée
    Dim numéro As Integer

    'Initialise le compteur de relations
    nombreRelations = 0
    'Pour toutes les classes d'attributs de la classe de parts ou d'états
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit de la relation recherchée
        If (tabParts(num, i, 0) Like relation) Then
            nombreRelations = nombreRelations + 1
        End If
        'Comptabilise le nombre de relations que la super classe possède
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            numéro = tabParts(num, i, 4)
            nombreRelations = nombreRelations + compterRelation(numéro, relation)
        End If
    Next i
    compterRelation = nombreRelations

End Function
```

F.7. Module standard "Module4"

```
'Module nommé "Module4". _
Traduit un modèle graphique de "parts" en langage AlbertII et place le code généré _
dans un fichier situé par défaut sur la racine "C:\". _
La procédure de ce module est appelée par la macro "traduireEnAlbertII" ("Module1").

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text

'Tableau à 2 dimensions qui contient les numéros des classes sous-ensembles feuilles _
(0,0) : contient le nombre de classes du tableau. _
(x,0) : contient le numéro d'une classe. _
(x,1) : contient un booléen indiquant si la classe est concernée par un type de l'énumération
Dim tabFeuilles(0 To MAXTAB, 0 To 1) As String
'Tableau à 2 dimensions qui contient les numéros des classes sous-ensembles noeuds _
(0,0) : contient le nombre de classes du tableau. _
(x,0) : contient le numéro d'une classe. _
(x,1) : contient un booléen indiquant si la classe est la super classe d'un type de l'énumération
Dim tabNoeuds(0 To MAXTAB, 0 To 1) As String
```

```
'Traduit le modèle graphique de "parts" et place le code généré dans un fichier

Public Sub enregistrerModèle()

    'Chemin et nom du fichier qui contiendra la traduction du modèle en AlbertII
    Dim chemin As String
    'Longueur d'une chaîne de caractères (nom du fichier AlbertII)
    Dim longueur As Integer
    'Nombre de parts et d'états présents dans le modèle graphique
    Dim nombreParts As Integer
    'Contient une classe de parts ou d'états traduite en langage AlbertII
    Dim classe As String
    'Booléen indiquant si un type de base est le premier à être imprimé dans le fichier
    Dim premierTypeBase As Boolean
    'Booléen indiquant si une classe de parts de base est un type de base
    Dim typeBase As Boolean
    'Numéro de la classe sous-ensemble à vérifier
    Dim numéro As Integer
    'Numéro de la super classe ou de la classe sous-ensemble à vérifier
    Dim numérobis As Integer
    'Booléen indiquant si un type construit est le premier à être imprimé dans le fichier
    Dim premierTypeConstruit As Boolean
    'Variable contenant une contrainte à imprimer
    Dim contrainte As String
    'Booléen indiquant si la contrainte d'un type de l'énumération est la première
    Dim premEnum As Boolean
```

```
'Compteur du nombre de pages
Dim x As Integer
'Compteur du nombre de parts et d'états présents dans le tableau temporaire
Dim i As Integer
'Compteur utilisé dans les tableaux des sous-classes feuilles et noeuds
Dim y As Integer
'Indice du tableau (dimension1) correspondant au numéro de la classe de parts ou d'états
Dim num As Integer
'Variable contenant la somme des bornes inférieures des relations dispositifs de fixation
Dim sommeBornesInf As Integer
'Variable contenant les contraintes temporaires
Dim temp As String
'Variable contenant le message adressé à l'utilisateur
Dim message As String
'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
Dim typeErreur As String
'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
Dim z As Integer

'Résolution en 5 temps : _
1) Ouvrir ou créer le fichier. _
2) Imprimer le titre du fichier. _
3) Imprimer les types de base. _
4) Imprimer les types construits. _
5) Fermer le fichier

'1) Ouverture (création) du fichier
'Valide la routine chargée de gérer une erreur sur le chemin d'accès au fichier
On Error GoTo erreur
'Envoie un message à l'utilisateur lui demandant le chemin et le nom du fichier
message = "Veuillez indiquer le chemin et le nom du fichier qui contiendra la traduction " _
    & "du modèle de ""parts"" en langage AlbertII." _
    & Chr(10) & _
    "N'oubliez pas de spécifier le lecteur ainsi que les répertoires et les sous-répertoires " _
    & "à parcourir pour localiser le fichier." _
    & Chr(10) & "Exemple : ""C:\documents\nom_du_fichier""." _
    & Chr(10) & Chr(10) & _
    "Si le fichier n'existe pas, il sera créé." _
    & Chr(10) & "Dans le cas contraire, son contenu sera écrasé." & Chr(10) & Chr(10)
'Affecte la valeur renvoyée de la fonction InputBox à chemin
chemin = InputBox(message, , "C:\Albert") & ".txt"
'Teste si le chemin est vide (notamment si l'utilisateur a appuyé sur le bouton "Annuler")
If (chemin Like ".txt") Then
    'Force l'arrêt de la macro "traduireEnAlbertII"
    vérification = False
    Exit Sub
End If
'Ouvre le fichier
Open chemin For Output As #1
'Evite la routine de gestion d'erreur
```

```
        GoTo toutvabien
erreur:
        'Affiche le message à l'utilisateur en cas d'échec
        message = "Le chemin d'accès au fichier est introuvable." _
                & Chr(10) & "Veuillez recommencer la traduction du modèle." & Chr(10) & Chr(10)
        typeErreur = "Erreur : chemin d'accès incorrect"
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
toutvabien:
        'Désactive la routine de gestion d'erreur
        On Error GoTo 0

        '2) Imprime le titre en langage AlbertII
        'Renvoie le nombre de caractères d'une chaîne (titre du projet ou du document)
        longueur = Len(ActiveDocument.Name)
        'Supprime l'extension ".vst" du titre du projet
        Print #1, "%SPEC " & Mid(ActiveDocument.Name, 1, longueur - 4)
        Print #1,

        '3) Imprime les types de base en langage AlbertII (+ commentaires)
        'Initialise le booléen qui signale le premier type de base à être enregistré
        premierTypeBase = True
        'Pour toutes les classes de parts et d'états
        For x = 1 To tabParts(0, 0, 0)
                'Renvoie le numéro de la classe de parts ou d'états
                num = tabParts(0, 0, x)
                'Initialise la variable booléen qui signale si la classe est un type de base
                typeBase = False
                'Le type de base est toujours ...
                Select Case tabParts(num, 0, 1)
                '... soit une classe d'états dispositifs simples;
                Case "PartPhysicalFeatureClass"
                        'Teste si la classe d'états dispositifs possède une classe d'attributs valeurs
                        If posséderRelation(num, "possible_values") Then
                                typeBase = False
                        Else
                                typeBase = True
                        End If
                '... soit une classe d'identités, de positions ou de dispositifs de valeur _
                dont le type est défini par l'utilisateur.
                Case "PartIdentityClass", "PartPositionClass", "PartFeatureValueClass"
                        'Teste si la classe d'états n'est pas un type prédéfini
                        If Not typePrédéfini(tabParts(num, 0, 2)) Then
                                typeBase = True
                        End If
                'Une classe de parts ou une classe de dispositifs géométriques, de fixation, _
                de contiguïté ou d'enclos n'est jamais un type de base
                Case "BasicPartClass", "CompoundPartClass", "BasicOrCompoundPartClass", "PileOfParts", _
```

```
            "Buffer", "Container", "PartGeometricalFeatureClass", "PartFixingFeatureClass", _
            "PartContiguityFeature", "PartEnclosureFeature"
        Case Else
            Debug.Print "La classe de parts ou d'états ne rentre pas dans la configuration prévue."
        End Select
        'Teste si la classe est un type de base
        If typeBase Then
            'Teste s'il s'agit du premier type de base à être enregistré
            If premierTypeBase Then
                'Imprime le mot-clé précédent les types de base
                Print #1, "%BASIC TYPES"
                Print #1,
                premierTypeBase = False
            End If
            'Imprime le type de base
            Print #1, tabParts(num, 0, 2)
            'Imprime les commentaires liés au type de base
            imprimerCommentaires num
            Print #1,
            'Signale que la classe de parts est un type de base
            tabParts(num, 0, 1) = "BasicType"
        End If
Next x

'4) Imprime les types construits en langage AlbertII (+ contraintes et commentaires)
'4.1) Imprime la formule qui précède la liste des types construits
'Renvoie le nombre de classes de parts et d'états présentes dans le tableau
nombreParts = tabParts(0, 0, 0)
'Teste si le type de base existe (il a donc été imprimé)
If Not (premierTypeBase) Then
    'Imprime un espace précédent les catégories de type construit
    Print #1,
End If
'Imprime le mot-clé précédent les types construits
Print #1, "%CONSTRUCTED TYPES"
Print #1,

'4.2) Imprime la liste des types construits
'A) Imprime la liste des classes de parts élémentaires (sans relation sous-ensemble)
'Initialise le booléen indiquant si un type construit est le premier à être imprimé
premierTypeConstruit = True
'Pour chaque classe de parts et d'états
For i = 1 To nombreParts
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste si la classe de parts ne possède pas de relation sous-ensemble
    If ((tabParts(num, 0, 1) Like "*PartClass") Or _
        (tabParts(num, 0, 1) Like "*PileOfParts") Or _
        (tabParts(num, 0, 1) Like "*Buffer") Or _
        (tabParts(num, 0, 1) Like "*Container")) And _
```

```
Not (posséderRelation(num, "isAOrigine")) And _
Not (posséderRelation(num, "isADestination")) Then
'Prépare le début de la traduction de la classe en langage AlbertII
classe = tabParts(num, 0, 2) & " = CP[type:" & Switch(tabParts(num, 0, 1) Like "*Class", _
    Mid(tabParts(num, 0, 1), 1, Len(tabParts(num, 0, 1)) - 5), True, tabParts(num, 0, 1)) _
    & "," & (Chr(13) & Chr(10)) & Chr(9)
'Traduit les classes d'attributs associées à la classe de parts
classe = classe & traduireClassesAttributs(num, True, False)
'Teste si le type construit est le premier de sa catégorie
If premierTypeConstruit Then
    'Imprime le commentaire précédent la catégorie de type construit
    Print #1, "//CLASSES DE PARTS ÉLÉMENTAIRES"
    Print #1,
    premierTypeConstruit = False
End If
'Imprime la traduction de la classe dans le fichier
Print #1, classe
'Imprime la contrainte d'identité dans le fichier
imprimerContrainteIdentité num
'Teste si la classe de parts est constituée de composants
If posséderRelation(num, "possible_components") Then
    'Enregistre l'éventuelle contrainte de composition
    contrainte = imprimerContrainteMultiple(num, "possible_components")
    'Teste si la contrainte doit être imprimée dans le fichier
    If Not (contrainte Like "") Then
        'Imprime la contrainte de composition dans le fichier
        Print #1, "    //Contrainte de composition"
        Print #1, "    \WITH \ForAll p/" & tabParts(num, 0, 2) & " : " & contrainte
    End If
End If
'Teste si la classe de parts est constituée de contenus
If posséderRelation(num, "contain") Then
    'Enregistre l'éventuelle contrainte de contenance
    contrainte = imprimerContrainteMultiple(num, "contain")
    'Teste si la contrainte doit être imprimée dans le fichier
    If Not (contrainte Like "") Then
        'Imprime la contrainte de contenance dans le fichier
        Print #1, "    //Contrainte de contenance"
        Print #1, "    \WITH \ForAll p/" & tabParts(num, 0, 2) & " : " & contrainte
    End If
End If
'Teste s'il existe une contrainte de cardinalité
contrainte = imprimerContrainteCardinalité(num, True, False)
If Not (contrainte Like "") Then
    'Imprime la contrainte de cardinalité dans le fichier
    Print #1, "    //Contrainte de cardinalité"
    Print #1, contrainte
End If
'Teste s'il s'agit d'une classe de piles ou de conteneurs
If posséderRelation(num, "contiguity") Then
```

```
                'Calcule la somme des bornes inférieures des relations contiguïtés
                sommeBornesInf = calculerSommeBornesInf(num, "contiguity")
                'Teste si la contrainte doit être imprimée dans le fichier
                If (sommeBornesInf = 0) Then
                    'Enregistre la contrainte de contiguïté
                    contrainte = traduireContrainteMultiple1(num, "contiguity", True)
                    'Imprime la contrainte de contiguïté dans le fichier
                    Print #1, "    //Contrainte de contiguïté"
                    Print #1, "    \WITH \ForAll p/" & tabParts(num, 0, 2) & " : " & contrainte
                End If
            End If
            'Imprime les commentaires
            imprimerCommentaires num
            Print #1,
        End If
    Next i
    'Teste si le type construit existe (il a donc été imprimé)
    If Not (premierTypeConstruit) Then
        'Imprime un espace précédent la catégorie de type construit suivante
        Print #1,
    End If
    'B) Imprime la liste des super classes de parts racines
    'Initialise le booléen indiquant si un type construit est le premier à être imprimé
    premierTypeConstruit = True
    'Pour chaque classe de parts et d'états
    For i = 1 To nombreParts
        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, i)
        'Teste si la classe de parts est uniquement la destination d'une relation sous-ensemble
        If Not (posséderRelation(num, "isAOrigine")) And _
            (posséderRelation(num, "isADestination")) Then
            'Prépare le début de la traduction de la classe en langage AlbertII
            classe = tabParts(num, 0, 2) & " = CP["
            'Traduit les classes d'attributs associées à la classe de parts
            classe = classe & traduireClassesAttributs(num, True, False)
            'Teste si le type construit est le premier de sa catégorie
            If premierTypeConstruit Then
                'Imprime le commentaire précédent la catégorie de type construit
                Print #1, "//SUPER CLASSES DE PARTS RACINES"
                Print #1,
                premierTypeConstruit = False
            End If
            'Imprime la traduction de la classe dans le fichier
            Print #1, classe
            'Imprime la contrainte d'identité dans le fichier
            imprimerContrainteIdentité num
            'Imprime les contraintes d'énumération, de composition et de cardinalité
            Print #1, "    //Contrainte d'énumération"
            'Enregistre le début de la contrainte
            contrainte = "    \WITH \ForAll p : " & tabParts(num, 0, 2)
```

```
'Pour toutes les classes feuilles
For x = 1 To tabFeuilles(0, 0)
    'Initialise le booléen indiquant si la contrainte d'un type _
    de l'énumération est la première
    premEnum = True
    'Initialise les tableaux de feuilles et de noeuds
    For y = 1 To tabFeuilles(0, 0)
        tabFeuilles(y, 1) = False
    Next y
    For y = 1 To tabNoeuds(0, 0)
        tabNoeuds(y, 1) = False
    Next y
    'Met à jour le tableau des feuilles
    tabFeuilles(x, 1) = True
    'Enregistre le numéro de la classe feuille qui fait l'objet de la contrainte
    numéro = tabFeuilles(x, 0)
    'Lance la procédure sélectionnant les super classes de la feuille
    sélectionnerSuperClasses numéro
    'Teste si le type n'est pas le premier
    If Not (x = 1) Then
        contrainte = contrainte & " \and"
    End If
    'Enregistre le début de la contrainte relative au type de l'énumération
    contrainte = contrainte & (Chr(13) & Chr(10)) & Chr(9) & "    " & _
        "(Type(p)=" & tabParts(numéro, 0, 2)
    'a) Enregistre les contraintes sur les classes d'attributs dispositifs, _
    composants, contenus, contenants, contiguïtés et enclos de la feuille
    temp = imprimerContrainteEnumération(numéro, premEnum, " <> ")
    If Not (temp Like "") Then
        contrainte = contrainte & temp
        premEnum = False
    End If
    'b) Enregistre les contraintes sur les classes d'attributs dispositifs, composants, _
    contenus, contenants, contiguïtés et enclos des super classes de la feuille _
    mais évite la classe racine
    For y = 2 To tabNoeuds(0, 0)
        'Teste si le noeud est une super classe de la feuille
        If (tabNoeuds(y, 1) Like True) Then
            'Enregistre le numéro de la super classe
            numérobis = tabNoeuds(y, 0)
            temp = imprimerContrainteEnumération(numérobis, premEnum, " <> ")
            If Not (temp Like "") Then
                contrainte = contrainte & temp
                premEnum = False
            End If
        End If
    Next y
    'c) Enregistre les contraintes sur les classes d'attributs dispositifs, _
    composants, contenus, contenants, contiguïtés et enclos des autres feuilles
    For y = 1 To tabFeuilles(0, 0)
```

```
        'Teste s'il ne s'agit pas de la feuille déjà enregistrée
        If (tabFeuilles(y, 1) Like False) Then
            'Enregistre le numéro de la super classe
            numérobis = tabFeuilles(y, 0)
            temp = imprimerContrainteEnumération(numérobis, premEnum, " = ")
            If Not (temp Like "") Then
                contrainte = contrainte & temp
                premEnum = False
            End If
        End If
    Next y
    'd) Enregistre les contraintes sur les classes d'attributs dispositifs, _
    composants, contenus, contenants, contiguïtés et enclos des autres super classes
    For y = 1 To tabNoeuds(0, 0)
        'Teste si la super classe n'a pas déjà été enregistrée
        If (tabNoeuds(y, 1) Like False) Then
            'Enregistre le numéro de la super classe
            numérobis = tabNoeuds(y, 0)
            temp = imprimerContrainteEnumération(numérobis, premEnum, " = ")
            If Not (temp Like "") Then
                contrainte = contrainte & temp
                premEnum = False
            End If
        End If
    Next y
    'e) Imprime la contrainte de composition
    'Teste si la feuille est constituée de composants
    If (tabParts(numéro, 0, 1) Like "CompoundPartClass") Or _
        (tabParts(numéro, 0, 1) Like "PileOfParts") Then
        'Teste si la contrainte de composition n'est pas vide
        temp = imprimerContrainteMultiple(numéro, "possible_components")
        If Not (temp Like "") Then
            'Teste si la contrainte est la première
            If Not (premEnum) Then
                contrainte = contrainte & " \and "
            Else
                premEnum = False
            End If
            'Enregistre la contrainte de composition
            contrainte = contrainte & temp
        End If
    End If
    'f) Imprime la contrainte de contenance
    'Teste si la feuille est constituée de contenus
    If (tabParts(numéro, 0, 1) Like "Container") Then
        'Teste si la contrainte de contenance n'est pas vide
        temp = imprimerContrainteMultiple(numéro, "contain")
        If Not (temp Like "") Then
            'Teste si la contrainte est la première
            If Not (premEnum) Then
```

```
                        contrainte = contrainte & " \and "
                    Else
                        premEnum = False
                    End If
                    'Enregistre la contrainte de contenance
                    contrainte = contrainte & temp
                End If
            End If
            'g) Imprime la contrainte de cardinalité
            contrainte = contrainte & imprimerContrainteCardinalité(numéro, premEnum, True)
            'h) Imprime la contrainte de contiguïté
            'Teste s'il s'agit d'une classe de piles ou de conteneurs
            If (tabParts(numéro, 0, 1) Like "PileOfParts") Or _
                (tabParts(numéro, 0, 1) Like "Container") Then
                'Calcule la somme des bornes inférieures des relations contiguïtés
                sommeBornesInf = calculerSommeBornesInf(numéro, "contiguity")
                'Teste si la contrainte doit être imprimée dans le fichier
                If (sommeBornesInf = 0) Then
                    'Teste si la contrainte est la première
                    If Not (premEnum) Then
                        contrainte = contrainte & " \and "
                    Else
                        premEnum = False
                    End If
                    'Enregistre la contrainte de contiguïté
                    contrainte = contrainte & traduireContrainteMultiple1(numéro, "contiguity", True)
                End If
            End If
            'Enregistre la fin de la contrainte relative au type
            contrainte = contrainte & ")"
        Next x
        'Imprime la contrainte
        Print #1, contrainte
        'Imprime les commentaires
        imprimerCommentaires num
        Print #1,
    End If
Next i
'Teste si le type construit existe (il a donc été imprimé)
If Not (premierTypeConstruit) Then
    'Imprime un espace précédent la catégorie de type construit suivante
    Print #1,
End If
'C) Imprime la liste des classes de parts à la fois sous-ensembles et super classes
'Initialise le booléen indiquant si un type construit est le premier à être imprimé
premierTypeConstruit = True
'Pour chaque classe de parts et d'états
For i = 1 To nombreParts
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
```

```
    'Teste si la classe de parts est un sous-ensemble et une super classe
    If posséderRelation(num, "isAOrigine") And posséderRelation(num, "isADestination") Then
        'Prépare la traduction de la classe en langage AlbertII
        classe = tabParts(num, 0, 2) & " = " & tabParts(tabParts(num, 1, 4), 0, 2)
        'Teste si le type construit est le premier de sa catégorie
        If premierTypeConstruit Then
            'Imprime le commentaire précédent la catégorie de type construit
            Print #1, "//CLASSES DE PARTS SOUS-ENSEMBLES ET SUPER CLASSES"
            Print #1,
            premierTypeConstruit = False
        End If
        'Imprime la traduction de la classe dans le fichier
        Print #1, classe
        'Imprime les commentaires
        imprimerCommentaires num
        Print #1,
    End If
Next i
'Teste si le type construit existe (il a donc été imprimé)
If Not (premierTypeConstruit) Then
    'Imprime un espace précédent la catégorie de type construit suivante
    Print #1,
End If
'D) Imprime la liste des classes de parts sous-ensembles feuilles
'Initialise le booléen indiquant si un type construit est le premier à être imprimé
premierTypeConstruit = True
'Pour chaque classe de parts et d'états
For i = 1 To nombreParts
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste si la classe de parts est un sous-ensemble feuille
    If posséderRelation(num, "isAOrigine") And Not (posséderRelation(num, "isADestination")) Then
        'Prépare la traduction de la classe en langage AlbertII
        classe = tabParts(num, 0, 2) & " = CP[type:" & Switch(tabParts(num, 0, 1) Like "*Class", _
            Mid(tabParts(num, 0, 1), 1, Len(tabParts(num, 0, 1)) - 5), True, tabParts(num, 0, 1)) _
            & "," & (Chr(13) & Chr(10)) & Chr(9) _
            & "isA:" & tabParts(tabParts(num, 1, 4), 0, 2) & "]"
        'Teste si le type construit est le premier de sa catégorie
        If premierTypeConstruit Then
            'Imprime le commentaire précédent la catégorie de type construit
            Print #1, "//SOUS-CLASSES DE PARTS FEUILLES"
            Print #1,
            premierTypeConstruit = False
        End If
        'Imprime la traduction de la classe dans le fichier
        Print #1, classe
        'Imprime les commentaires
        imprimerCommentaires num
        Print #1,
    End If
```

```
Next i
'Teste si le type construit existe (il a donc été imprimé)
If Not (premierTypeConstruit) Then
    'Imprime un espace précédent la catégorie de type construit suivante
    Print #1,
End If
'E) Imprime la liste des classes de dispositifs physiques
'Initialise le booléen indiquant si un type construit est le premier à être imprimé
premierTypeConstruit = True
'Pour chaque classe de parts et d'états
For i = 1 To nombreParts
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste si la classe de parts est un dispositif physique
    If (tabParts(num, 0, 1) Like "PartPhysicalFeatureClass") Then
        'Prépare le début de la traduction de la classe en langage AlbertII
        classe = tabParts(num, 0, 2) & " = CP[type:PartPhysicalFeature," _
            & (Chr(13) & Chr(10)) & Chr(9)
        'Traduit les classes d'attributs associées à la classe de dispositifs physiques
        classe = classe & traduireClassesAttributs(num, True, False)
        'Teste si le type construit est le premier de sa catégorie
        If premierTypeConstruit Then
            'Imprime le commentaire précédent la catégorie de type construit
            Print #1, "//CLASSES DE DISPOSITIFS PHYSIQUES"
            Print #1,
            premierTypeConstruit = False
        End If
        'Imprime la traduction de la classe dans le fichier
        Print #1, classe
        'Teste s'il existe une contrainte de cardinalité
        contrainte = imprimerContrainteCardinalité(num, True, False)
        If Not (contrainte Like "") Then
            'Imprime la contrainte de cardinalité dans le fichier
            Print #1, "    //Contrainte de cardinalité"
            Print #1, contrainte
        End If
        'Imprime les commentaires
        imprimerCommentaires num
        Print #1,
    End If
Next i
'Teste si le type construit existe (il a donc été imprimé)
If Not (premierTypeConstruit) Then
    'Imprime un espace précédent la catégorie de type construit suivante
    Print #1,
End If
'F) Imprime la liste des classes de dispositifs géométriques
'Initialise le booléen indiquant si un type construit est le premier à être imprimé
premierTypeConstruit = True
'Pour chaque classe de parts et d'états
```

```
For i = 1 To nombreParts
    'Renvoie le numéro de la classe de parts ou d'états
    num = tabParts(0, 0, i)
    'Teste si la classe de parts est un dispositif géométrique
    If (tabParts(num, 0, 1) Like "PartGeometricalFeatureClass") Then
        'Prépare le début de la traduction de la classe en langage AlbertII
        classe = tabParts(num, 0, 2) & " = CP[type:PartGeometricalFeature," _
            & (Chr(13) & Chr(10)) & Chr(9)
        'Traduit les classes d'attributs associées à la classe de dispositifs géométriques
        classe = classe & traduireClassesAttributs(num, True, False)
        'Teste si le type construit est le premier de sa catégorie
        If premierTypeConstruit Then
            'Imprime le commentaire précédent la catégorie de type construit
            Print #1, "//CLASSES DE DISPOSITIFS GÉOMÉTRIQUES"
            Print #1,
            premierTypeConstruit = False
        End If
        'Imprime la traduction de la classe dans le fichier
        Print #1, classe
        'Teste s'il existe une contrainte de cardinalité
        contrainte = imprimerContrainteCardinalité(num, True, False)
        If Not (contrainte Like "") Then
            'Imprime la contrainte de cardinalité dans le fichier
            Print #1, "    //Contrainte de cardinalité"
            Print #1, contrainte
        End If
        'Enregistre l'éventuelle contrainte de géométrie
        contrainte = imprimerContrainteMultiple(num, "possible_geometrical_features")
        'Teste si la contrainte doit être imprimée dans le fichier
        If Not (contrainte Like "") Then
            'Imprime la contrainte de géométrie dans le fichier
            Print #1, "    //Contrainte de géométrie"
            Print #1, "    \WITH \ForAll p/" & tabParts(num, 0, 2) & " : " & contrainte
        End If
        'Imprime les commentaires
        imprimerCommentaires num
        Print #1,
    End If
Next i
'Teste si le type construit existe (il a donc été imprimé)
If Not (premierTypeConstruit) Then
    'Imprime un espace précédent la catégorie de type construit suivante
    Print #1,
End If
'G) Imprime la liste des classes de dispositifs de fixation
'Initialise le booléen indiquant si un type construit est le premier à être imprimé
premierTypeConstruit = True
'Pour chaque classe de parts et d'états
For i = 1 To nombreParts
    'Renvoie le numéro de la classe de parts ou d'états
```

```
        num = tabParts(0, 0, i)
        'Teste si la classe de parts est un dispositif de fixation
        If (tabParts(num, 0, 1) Like "PartFixingFeatureClass") Then
            'Prépare le début de la traduction de la classe en langage AlbertII
            classe = tabParts(num, 0, 2) & " = CP[type:PartFixingFeature," _
                & (Chr(13) & Chr(10)) & Chr(9)
            'Traduit les classes d'attributs associées à la classe de dispositifs de fixation
            classe = classe & traduireClassesAttributs(num, True, False)
            'Teste si le type construit est le premier de sa catégorie
            If premierTypeConstruit Then
                'Imprime le commentaire précédent la catégorie de type construit
                Print #1, "//CLASSES DE DISPOSITIFS DE FIXATION"
                Print #1,
                premierTypeConstruit = False
            End If
            'Imprime la traduction de la classe dans le fichier
            Print #1, classe
            'Teste s'il existe une contrainte de cardinalité
            contrainte = imprimerContrainteCardinalité(num, True, False)
            If Not (contrainte Like "") Then
                'Imprime la contrainte de cardinalité dans le fichier
                Print #1, "    //Contrainte de cardinalité"
                Print #1, contrainte
            End If
            'Calcule la somme des bornes inférieures des relations dispositifs de fixation
            sommeBornesInf = calculerSommeBornesInf(num, "possible_fixing_features")
            'Teste si la contrainte doit être imprimée dans le fichier
            If (sommeBornesInf = 0) Then
                'Enregistre la contrainte de fixation
                contrainte = traduireContrainteMultiple1(num, "possible_fixing_features", True)
                'Imprime la contrainte de fixation dans le fichier
                Print #1, "    //Contrainte de fixation"
                Print #1, "    \WITH \ForAll p/" & tabParts(num, 0, 2) & " : " & contrainte
            End If
            'Imprime les commentaires
            imprimerCommentaires num
            Print #1,
        End If
    Next i
    'Teste si le type construit existe (il a donc été imprimé)
    If Not (premierTypeConstruit) Then
        'Imprime un espace précédent la catégorie de type construit suivante
        Print #1,
    End If
    'H) Imprime la liste des classes de dispositifs de contiguïté
    'Initialise le booléen indiquant si un type construit est le premier à être imprimé
    premierTypeConstruit = True
    'Pour chaque classe de parts et d'états
    For i = 1 To nombreParts
        'Renvoie le numéro de la classe de parts ou d'états
```

```
        num = tabParts(0, 0, i)
        'Teste si la classe de parts est un dispositif de contiguïté
        If (tabParts(num, 0, 1) Like "PartContiguityFeature") Then
            'Prépare le début de la traduction de la classe en langage AlbertII
            classe = tabParts(num, 0, 2) & " = CP[type:PartContiguityFeature," _
                & (Chr(13) & Chr(10)) & Chr(9)
            'Traduit les classes d'attributs associées à la classe de dispositifs de contiguïté
            classe = classe & traduireClassesAttributs(num, True, False)
            'Teste si le type construit est le premier de sa catégorie
            If premierTypeConstruit Then
                'Imprime le commentaire précédent la catégorie de type construit
                Print #1, "//CLASSES DE DISPOSITIFS DE CONTIGUÏTÉ"
                Print #1,
                premierTypeConstruit = False
            End If
            'Imprime la traduction de la classe dans le fichier
            Print #1, classe
            'Teste s'il existe une contrainte de cardinalité
            contrainte = imprimerContrainteCardinalité(num, True, False)
            If Not (contrainte Like "") Then
                'Imprime la contrainte de cardinalité dans le fichier
                Print #1, "    //Contrainte de cardinalité"
                Print #1, contrainte
            End If
            'Imprime les commentaires
            imprimerCommentaires num
            Print #1,
        End If
    Next i
    'Teste si le type construit existe (il a donc été imprimé)
    If Not (premierTypeConstruit) Then
        'Imprime un espace précédent la catégorie de type construit suivante
        Print #1,
    End If
    'I) Imprime la liste des classes de dispositifs d'enclos
    'Initialise le booléen indiquant si un type construit est le premier à être imprimé
    premierTypeConstruit = True
    'Pour chaque classe de parts et d'états
    For i = 1 To nombreParts
        'Renvoie le numéro de la classe de parts ou d'états
        num = tabParts(0, 0, i)
        'Teste si la classe de parts est un dispositif d'enclos
        If (tabParts(num, 0, 1) Like "PartEnclosureFeature") Then
            'Prépare le début de la traduction de la classe en langage AlbertII
            classe = tabParts(num, 0, 2) & " = CP[type:PartEnclosureFeature," _
                & (Chr(13) & Chr(10)) & Chr(9)
            'Traduit les classes d'attributs associées à la classe de dispositifs d'enclos
            classe = classe & traduireClassesAttributs(num, True, False)
            'Teste si le type construit est le premier de sa catégorie
            If premierTypeConstruit Then
```

```
                'Imprime le commentaire précédent la catégorie de type construit
                Print #1, "//CLASSES DE DISPOSITIFS D'ENCLOS"
                Print #1,
                premierTypeConstruit = False
            End If
            'Imprime la traduction de la classe dans le fichier
            Print #1, classe
            'Teste s'il existe une contrainte de cardinalité
            contrainte = imprimerContrainteCardinalité(num, True, False)
            If Not (contrainte Like "") Then
                'Imprime la contrainte de cardinalité dans le fichier
                Print #1, "    //Contrainte de cardinalité"
                Print #1, contrainte
            End If
            'Imprime les commentaires
            imprimerCommentaires num
            Print #1,
        End If
    Next i

    '5) Fermeture du fichier
    Close #1

End Sub
```

```
'Traduit en langage AlbertII les classes d'attributs d'une classe de parts ou d'états
'num : numéro associé à la classe de parts ou d'états
'prem : booléen signalant si la classe d'attributs est la première dans la liste
'résursivité : booléen signalant si la fonction est exécutée suite à un appel récursif

Private Function traduireClassesAttributs(num As Integer, prem As Boolean, _
    récursivité As Boolean)

    'Variable contenant la traduction de la classe de parts ou d'états
    Dim classe As String
    'Variable contenant la traduction en langage AlbertII d'une classe d'attributs
    Dim attribut As String
    'Variable contenant le numéro de la classe de parts ou d'états à traduire
    Dim numéro As Integer
    'Booléen indiquant si la relation est la première dans la liste (pour placer la virgule)
    Dim premier As Boolean
    'Booléen indiquant si la fonction est exécutée suite à un appel récursif
    Dim récursif As Boolean
    'Booléen indiquant si l'énumération des classes sous-ensembles a été imprimée
    Dim premEnum As Boolean
    'Compteur de classes d'attributs
    Dim i, x As Integer
    'Variables contenant le commentaire à imprimer
    Dim commentaire As String, commentairebis As String
```

```
'Initialise les variables
premier = prem
récursif = récursivité
commentaire = ""
commentairebis = ""
'Initialise le booléen de l'énumération
premEnum = True

'Pour toutes les classes d'attributs
For i = 1 To tabParts(num, 0, 0)
    'Initialise la variable qui contiendra la traduction de la classe d'attributs
    attribut = ""
    'Traitement de la classe d'attributs selon sa méta-classe : traduction ou autre
    Select Case tabParts(num, i, 0)
    'Classes d'attributs "simples"
    Case "possible_components", "possible_contents", "contain", "possible_identities", _
        "possible_positions", "possible_physical_features", "possible_geometry", _
        "possible_geometrical_features", "possible_fixing_features", "contiguity_feature", _
        "enclosure_feature", "contiguity", "enclosure", "possible_values"
        attribut = attribut & tabParts(num, i, 1) & ":"
        'Teste si la cardinalité renvoie à une valeur
        If (tabParts(num, i, 3) Like "1") Then
            attribut = attribut & tabParts(tabParts(num, i, 4), 0, 2)
            'Teste si la classe d'attributs est facultative ("0-1" ou sous-ensemble)
            If (tabParts(num, i, 2) Like "0") Or récursif Then
                attribut = attribut & "*"
            End If
        Else
            'Teste si la cardinalité renvoie à un ensemble
            If (tabParts(num, i, 5) Like "FAUX") Or _
                Not (tabParts(num, i, 0) Like "possible_values") Then
                attribut = attribut & "SET[" & tabParts(tabParts(num, i, 4), 0, 2) & "]"
            End If
            'Teste si la cardinalité renvoie à une séquence
            If (tabParts(num, i, 6) Like "VRAI") Then
                attribut = attribut & "SEQ[" & tabParts(tabParts(num, i, 4), 0, 2) & "]"
            End If
            'Teste si la cardinalité renvoie à un sac
            If (tabParts(num, i, 5) Like "VRAI") And (tabParts(num, i, 6) Like "FAUX") Then
                attribut = attribut & "BAG[" & tabParts(tabParts(num, i, 4), 0, 2) & "]"
            End If
        End If
        'Ajoute le commentaire lié à la classe d'identités, de positions ou de dispositifs de _
        valeur dont le nom est un type prédéfini (sinon commentaire imprimé dans types de base)
        If (tabParts(tabParts(num, i, 4), 0, 1) Like "PartIdentityClass") Or _
            (tabParts(tabParts(num, i, 4), 0, 1) Like "PartPositionClass") Or _
            (tabParts(tabParts(num, i, 4), 0, 1) Like "PartFeatureValueClass") And _
            typePrédéfini(tabParts(tabParts(num, i, 4), 0, 2)) Then
            'Ajoute le commentaire lié à la classe
```

```
            For x = 1 To tabParts(tabParts(num, i, 4), 0, 0)
                If (tabParts(tabParts(num, i, 4), x, 0) Like "Comment") Then
                    commentairebis = commentairebis & (Chr(13) & Chr(10)) & Chr(9) & _
                        "//" & tabParts(tabParts(num, i, 4), x, 1)
                End If
            Next x
        End If
    'Classe d'attributs nécessitant une énumération et l'emploi de la récursivité
    Case "isADestination"
        'Teste si l'énumération de la racine (!) n'a pas encore été traduite
        If premEnum And Not (récursif) Then
            'Initialise les tableaux contenant les feuilles et les noeuds
            tabFeuilles(0, 0) = 0
            tabNoeuds(0, 0) = 1
            tabNoeuds(1, 0) = num
            'Enregistre l'énumération des classes de parts sous-ensembles feuilles
            attribut = "type:ENUM[" & chercherFeuilles(num, True) & "]"
            'Indique que l'énumération a été enregistrée
            premEnum = False
            'ATTENTION : oblige la fonction à repasser une deuxième fois la première _
            des classes sous-ensembles afin de traduire ses classes d'attributs ("else")
            i = i - 1
        Else
            'Traduit les classes d'attributs de la classe sous-ensemble
            numéro = tabParts(num, i, 4)
            attribut = traduireClassesAttributs(numéro, True, True)
        End If
    'Les cas de la sous-classe et du commentaire sont traités dans la procédure appelante
    Case "isAOrigine", "Comment"
    Case Else
        Debug.Print "La classe d'attributs ne rentre pas dans la configuration prévue."
    End Select
    'Teste s'il existe une classe d'attributs à imprimer
    If Not (attribut Like "") Then
        'Teste si la relation est la première (donc pas précédée par une virgule)
        If premier Then
            'Teste si le commentaire est vide
            If (commentaire Like "") Then
                'Imprime la nouvelle classe d'états
                classe = attribut
            Else
                'Imprime le commentaire de la classe d'états précédente
                classe = commentaire & (Chr(13) & Chr(10)) & Chr(9) & attribut
            End If
            'Met à jour la variable indiquant si l'attribut est le premier dans la liste
            premier = False
        Else
            'Imprime le commentaire de la classe d'états précédente
            classe = classe & "," & commentaire
            'Imprime la nouvelle classe d'états
```

```
                classe = classe & (Chr(13) & Chr(10)) & Chr(9) & attribut
            End If
            'Enregistre le commentaire de la nouvelle classe d'états
            commentaire = commentairebis
            'Initialise la variable qui contiendra le commentaire de la prochaine classe d'états
            commentairebis = ""
        End If
    Next i
    'Teste si la fonction est appelée par la macro "traduireEnAlbertII"
    If Not (récursif) Then
        'Prépare la fin de la traduction de la classe en langage AlbertII
        classe = classe & "]" & commentaire
    End If
    traduireClassesAttributs = classe

End Function
```

```
'Renvoi les sous-classes feuilles de la super classe de parts racine
'num : numéro associé à la super classe de parts racine
'premier : booléen signalant si le sous-ensemble est le premier dans l'énumération

Private Function chercherFeuilles(num As Integer, premier)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Booléen indiquant si le sous-ensemble est le premier dans l'énumération (placer la virgule)
    Dim prem As Boolean
    'Variable contenant l'énumération des sous-classes feuilles
    Dim feuilles As String
    'Variable contenant le numéro d'une sous-classe
    Dim numéro As Integer

    'initialise le booléen
    prem = premier
    'Pour toutes les classes d'attributs
    For i = 1 To tabParts(num, 0, 0)
        'Teste si la classe d'attributs renvoie à un sous-ensemble
        If (tabParts(num, i, 0) Like "isADestination") Then
            'Renvoie le numéro de la sous-classe
            numéro = tabParts(num, i, 4)
            'Teste si la sous-classe est une feuille
            If Not (posséderRelation(numéro, "isADestination")) Then
                'Enregistre la sous-classe dans le tableau des feuilles
                tabFeuilles(0, 0) = tabFeuilles(0, 0) + 1
                tabFeuilles(tabFeuilles(0, 0), 0) = numéro
                'Enregistre la sous-classe
                If prem Then
                    feuilles = tabParts(numéro, 0, 2)
                    prem = False
```

```
                Else
                    feuilles = feuilles & ", " & tabParts(numéro, 0, 2)
                End If
            'La sous-classe est aussi une super classe
            Else
                'Enregistre la sous-classe dans le tableau des noeuds
                tabNoeuds(0, 0) = tabNoeuds(0, 0) + 1
                tabNoeuds(tabNoeuds(0, 0), 0) = numéro
                'Lance la recherche de ses sous-classes feuilles
                feuilles = feuilles & chercherFeuilles(numéro, prem)
            End If
        End If
    Next i
    chercherFeuilles = feuilles

End Function
```

```
'Imprime la contrainte relative aux classes d'identité pour la classe de parts n° "num"
'num : numéro de la classe de parts

Private Sub imprimerContrainteIdentité(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Booléen indiquant si la classe d'attributs identités est la première
    Dim prem As Boolean
    'Variable contenant la traduction de la contrainte d'identité
    Dim identité As String

    'Initialise les variables
    identité = "    \WITH \ForAll p/" & tabParts(num, 0, 2) & " \and \ForAll q/" _
        & tabParts(num, 0, 2) & " : p <> q => "
    prem = True
    'Pour toutes les classes d'attributs
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs identités
        If (tabParts(num, i, 0) Like "possible_identities") Then
            'Teste si la classe n'est pas la première
            If Not (prem) Then
                identité = identité & " \or "
            End If
            'Enregistre la contrainte liée à la classe d'attributs identités
            identité = identité & tabParts(num, i, 1) & _
                "(p) <> " & tabParts(num, i, 1) & "(q)"
            prem = False
        End If
    Next i
    'Imprime la contrainte d'identité dans le fichier
    Print #1, "    //Contrainte d'identité"
```

```
    Print #1, identité

End Sub
```

```
'Renvoie la contrainte relative aux composants pour la classe de parts n° "num"
'Renvoie la contrainte relative aux contenus pour la classe de conteneurs n° "num"
'Renvoie la contrainte relative aux géométries pour la classe de dispositifs géométriques n° "num"
'num : numéro de la classe de parts composées, de piles ou de dispositifs géométriques
'relation : type de relation (composant ou dispositif géométrique)

Private Function imprimerContrainteMultiple(num As Integer, relation As String)

    'Variable contenant la somme des bornes inférieures des relations
    Dim sommeBornesInf As Integer
    'Variable contenant la traduction de la contrainte
    Dim composant As String

    'Initialise la variable indiquant la somme des bornes inférieures
    sommeBornesInf = 0
    'Initialise la variable contenant la contrainte
    composant = ""
    'Calcule la somme des bornes inférieures des relations
    sommeBornesInf = calculerSommeBornesInf(num, relation)
    'Teste si la somme des bornes inférieures nécessite l'impression d'une contrainte
    If (sommeBornesInf = 0) Then
        'Lance la traduction de la contrainte
        composant = traduireContrainteMultiple0(num, relation, True, False)
    End If
    If (sommeBornesInf = 1) Then
        'Lance la traduction de la contrainte
        composant = traduireContrainteMultiple1(num, relation, True)
    End If
    imprimerContrainteMultiple = composant

End Function
```

```
'Renvoie la somme des bornes inférieures des relations "relation" de la classe n° "num"
'num : numéro de la classe de parts composées, de piles ou de dispositifs géométriques
'relation : type de relation (composant ou dispositif géométrique)

Private Function calculerSommeBornesInf(num As Integer, relation As String)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Variable contenant la somme des bornes inférieures des relations
    Dim sommeBornesInf As Integer
    'Variable contenant le numéro d'une super classe de parts ou d'une classe de dispositifs _
    géométriques dont on doit additionner les bornes inférieures des relations
```

```
    Dim numéro As Integer

    'Initialise la variable contenant la somme des bornes inférieures
    sommeBornesInf = 0
    'Pour toutes les relations
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une relation concernée
        If (tabParts(num, i, 0) Like relation) Then
            'Comptabilise les bornes inférieures
            sommeBornesInf = sommeBornesInf + tabParts(num, i, 2)
        End If
        'Teste si la classe est un sous-ensemble (nécessitant la récursivité)
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            numéro = tabParts(num, i, 4)
            sommeBornesInf = sommeBornesInf + calculerSommeBornesInf(numéro, relation)
        End If
    Next i
    calculerSommeBornesInf = sommeBornesInf

End Function
```

```
'Renvoie la contrainte de composition relative à la classe de parts n° "num"
'Renvoie la contrainte de géométrie relative à la classe de dispositifs géométriques n° "num"
'num : numéro de la classe de parts composées, de piles ou de dispositifs géométriques
'relation : type de relation (composant ou dispositif géométrique)
'premier : booléen indiquant si la relation est la première dans la contrainte
'récursif : booléen indiquant si la fonction est dans un cycle récursif

Private Function traduireContrainteMultiple0(num As Integer, relation As String, _
    premier As Boolean, récursif As Boolean)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Variable contenant la traduction de la contrainte
    Dim composant As String
    'Booléen indiquant si la classe d'attributs "relation" est la première
    Dim prem As Boolean
    'Variable contenant le numéro d'une super classe de parts ou d'une classe de dispositifs _
    géométriques dont on doit additionner les bornes inférieures des relations
    Dim numéro As Integer

    'Intialise le booléen indiquant si la classe d'attributs est la première
    prem = premier
    'Initialise la variable contenant la traduction de la contrainte
    If récursif Then
        composant = ""
    Else
        'Prépare le début de la traduction de la contrainte en langage AlbertII
        composant = "("
```

```
    End If
    'Enregistre la contrainte relative à la classe d'attributs "relation"
    For i = tabParts(num, 0, 0) To 1 Step -1
        'Teste s'il s'agit d'une relation concernée
        If (tabParts(num, i, 0) Like relation) Then
            'Teste si la classe n'est pas la première
            If Not (prem) Then
                composant = composant & " + "
            End If
            'Enregistre la contrainte liée à la classe d'attributs "relation"
            composant = composant & "Card(" & tabParts(num, i, 1) & ")"
            prem = False
        End If
        'Teste s'il s'agit d'une classe sous-ensemble (nécessitant la récursivité)
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            numéro = tabParts(num, i, 4)
            composant = composant & traduireContrainteMultiple0(numéro, relation, prem, True)
        End If
    Next i
    'Teste si la fonction est dans un cycle récursif
    If Not (récursif) Then
        'Prépare la fin de la traduction de la contrainte en langage AlbertII
        composant = composant & ") >= 2"
    End If
    traduireContrainteMultiple0 = composant

End Function
```

```
'Renvoie la contrainte de composition relative à la classe de parts n° "num"
'Renvoie la contrainte de géométrie relative à la classe de dispositifs géométriques n° "num"
'num : numéro de la classe de parts composées, de piles ou de dispositifs géométriques
'relation : type de relation (composant ou dispositif géométrique)
'premier : booléen indiquant si la relation est la première dans la contrainte

Private Function traduireContrainteMultiple1(num As Integer, relation As String, premier As Boolean)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Variable contenant la traduction de la contrainte
    Dim composant As String
    'Booléen indiquant si la classe d'attributs "relation" est la première
    Dim prem As Boolean
    'Variable contenant le numéro d'une super classe de parts ou d'une classe de dispositifs _
    géométriques dont on doit additionner les bornes inférieures des relations
    Dim numéro As Integer

    'Intialise le booléen indiquant si la classe d'attributs est la première
    prem = premier
    'Initialise la variable contenant la traduction de la contrainte
```

```
    composant = ""
    'Enregistre la contrainte relative à la classe d'attributs "relation"
    For i = tabParts(num, 0, 0) To 1 Step -1
        'Teste s'il s'agit d'une relation "relation"
        If (tabParts(num, i, 0) Like relation) Then
            'Teste si les deux bornes sont différentes (pas "1:1")
            If Not (tabParts(num, i, 2) Like tabParts(num, i, 3)) Then
                'Teste si la classe n'est pas la première
                If Not (prem) Then
                    composant = composant & " \or "
                End If
                'Cas 1 : "0:N" ou "1:N"
                If (tabParts(num, i, 3) Like "N") Then
                    composant = composant & "Card(" & tabParts(num, i, 1) & ") >= " _
                        & tabParts(num, i, 2) + 1
                Else
                    'Cas 2 : "0:1" ou "1:2"
                    If (tabParts(num, i, 3) = tabParts(num, i, 2) + 1) Then
                        composant = composant & "Card(" & tabParts(num, i, 1) & ") = " _
                            & tabParts(num, i, 3)
                    'Cas 3 : "0:2", "0:3", ... ou "1:3", "1:4", ...
                    Else
                        composant = composant & "Card(" & tabParts(num, i, 1) & ") >= " _
                            & tabParts(num, i, 2) + 1
                    End If
                End If
                prem = False
            End If
        End If
        'Teste s'il s'agit d'une classe sous-ensemble (nécessitant la récursivité)
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            numéro = tabParts(num, i, 4)
            composant = composant & traduireContrainteMultiple1(numéro, relation, prem)
        End If
    Next i
    traduireContrainteMultiple1 = composant

End Function
```

```
'Imprime la contrainte relative aux cardinalités des classes d'attributs de la classe n° "num"
'num : numéro de la classe de parts ou d'états
'premier : booléen indiquant si la classe d'attributs est la première
'sousEnsemble : booléen indiquant si la fonction est appelée par un sous-ensemble

Private Function imprimerContrainteCardinalité(num As Integer, premier As Boolean, _
    sousEnsemble As Boolean)

    'Compteur de classes d'attributs
    Dim i As Integer
```

```
'Variable contenant la traduction de la contrainte de cardinalité
Dim cardinalité As String
'Variable contenant le numéro de la classe de parts ou d'états dont la cardinalité _
des classes d'attributs doit être vérifiée
Dim numéro As Integer
'Borne inférieure de la cardinalité
Dim borneInf As String
'Borne supérieure de la cardinalité
Dim borneSup As String
'Booléen indiquant si la classe d'attributs est la première
Dim prem As Boolean

'Intialise le booléen indiquant si la classe d'attributs est la première
prem = premier
'Initialise la contrainte
cardinalité = ""
'Pour toutes les classes d'attributs
For i = tabParts(num, 0, 0) To 1 Step -1
    'Vérifie si une contrainte existe sur la cardinalité de la classe d'attributs
    Select Case tabParts(num, i, 0)
    'Classe d'attributs dont la cardinalité peut nécessiter une contrainte
    Case "possible_components", "possible_physical_features", "possible_geometrical_features", _
        "possible_fixing_features", "possible_contents", "contiguity_feature", _
        "enclosure_feature", "contiguity", "possible_values", "possible_geometry", "contain"
        'Initialise les variables contenant les bornes
        borneInf = tabParts(num, i, 2)
        borneSup = tabParts(num, i, 3)
        'Teste si les bornes de la cardinalité sont des entiers
        If IsNumeric(borneInf) And IsNumeric(borneSup) Then
            'Teste une contrainte du type "0:2"
            If (borneInf Like "0") And (borneSup > 1) Then
                cardinalité = cardinalité & enregistrerContrainteCardinalité(tabParts(num, 0, 2), _
                    tabParts(num, i, 1), "<=", tabParts(num, i, 3), prem, sousEnsemble)
                prem = False
            End If
            'Teste une contrainte du type "1:2"
            If (Cint(borneInf) < Cint(borneSup)) And Not (borneInf Like "0") Then
                cardinalité = cardinalité & enregistrerContrainteCardinalité(tabParts(num, 0, 2), _
                    tabParts(num, i, 1), ">=", tabParts(num, i, 2), prem, sousEnsemble)
                prem = False
                cardinalité = cardinalité & enregistrerContrainteCardinalité(tabParts(num, 0, 2), _
                    tabParts(num, i, 1), "<=", tabParts(num, i, 3), prem, sousEnsemble)
            End If
            'Teste une contrainte du type "2:2"
            If (borneInf Like borneSup) And (borneInf > 1) Then
                cardinalité = cardinalité & enregistrerContrainteCardinalité(tabParts(num, 0, 2), _
                    tabParts(num, i, 1), "=", tabParts(num, i, 2), prem, sousEnsemble)
                prem = False
            End If
        Else
```

```
                    'Teste une contrainte du type "1:N"
                    If (borneSup Like "N") And Not (borneInf Like "0") _
                        And Not (borneInf Like "N") Then
                        cardinalité = cardinalité & enregistrerContrainteCardinalité(tabParts(num, 0, 2), _
                            tabParts(num, i, 1), ">=", tabParts(num, i, 2), prem, sousEnsemble)
                        prem = False
                    End If
                End If
            'Classe d'attributs nécessitant l'emploi de la récursivité
            Case "isAOrigine"
                numéro = tabParts(num, i, 4)
                cardinalité = cardinalité & imprimerContrainteCardinalité(numéro, prem, sousEnsemble)
            'Classes d'attributs dont la cardinalité ne nécessite pas de contrainte
            Case "possible_identities", "possible_positions", "enclosure", "isADestination", "Comment"
            Case Else
                Debug.Print "La classe d'attributs ne rentre pas dans la configuration prévue."
            End Select
        Next i
        imprimerContrainteCardinalité = cardinalité

End Function
```

```
'Renvoie la contrainte relative à la cardinalité d'une classe d'attributs
'classe : classe de parts ou d'états dispositifs dont on teste les classes d'attributs
'attribut : classe d'attributs sur laquelle s'exerce une contrainte de cardinalité
'opérateur : opérateur de comparaison de la contrainte de cardinalité
'prem : booléen indiquant si la contrainte est la première
'sous-Ensemble : booléen indiquant si la classe de parts ou d'états est un sous-ensemble

Private Function enregistrerContrainteCardinalité(classe As String, attribut As String, _
    opérateur As String, num As String, prem As Boolean, sousEnsemble As Boolean)

    'Teste si la classe de parts ou d'attributs est un sous-ensemble
    If sousEnsemble Then
        'Teste si la contrainte est la première parmi les contraintes sur la classe _
        de parts sous-ensemble feuille
        If prem Then
            enregistrerContrainteCardinalité = " <=> Card(" & attribut & ") " & _
                opérateur & " " & num
        Else
            enregistrerContrainteCardinalité = " \and Card(" & attribut & ") " & _
                opérateur & " " & num
        End If
    Else
        'Teste si la contrainte est la première parmi les contraintes sur la cardinalité _
        des classes d'attributs d'une classe de parts ou d'états
        If prem Then
            enregistrerContrainteCardinalité = "    \WITH \ForAll p/" & classe & " : Card(" & _
                attribut & ") " & opérateur & " " & num
```

```
        Else
            enregistrerContrainteCardinalité = (Chr(13) & Chr(10)) & Chr(9) & _
                "    \and Card(" & attribut & ") " & opérateur & " " & num
        End If
    End If

End Function
```

```
'Imprime les commentaires de la classe n° "num"
'num : numéro de la classe de parts ou d'états

Private Sub imprimerCommentaires(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer

    'Pour toutes les classes d'attributs
    For i = 1 To tabParts(num, 0, 0)
        'Teste si la classe d'attributs est un commentaire
        If (tabParts(num, i, 0) Like "Comment") Then
            'Imprime le commentaire
            Print #1, "    //" & tabParts(num, i, 1)
        End If
    Next i

End Sub
```

```
'Sélectionne les super classes de la classe feuille n° "num" dans le tableau "tabFeuilles"
'num : numéro de la sous-classe feuille

Private Sub sélectionnerSuperClasses(num As Integer)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Compteur de classes noeuds
    Dim x As Integer
    'Numéro de la classe dont on cherche les éventuelles super classes
    Dim numéro As Integer

    'Pour toutes les classes d'attributs
    For i = 1 To tabParts(num, 0, 0)
        'Teste si la classe d'attributs renvoie à une super classe
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            'Enregistre le numéro de la super classe
            numéro = tabParts(num, i, 4)
            'Met à jour le tableau des noeuds
            For x = 1 To tabNoeuds(0, 0)
                'Teste s'il s'agit de la classe noeud sélectionnée
```

```
                If (numéro Like tabNoeuds(x, 0)) Then
                    tabNoeuds(x, 1) = True
                End If
            Next x
            'Poursuit la sélection au niveau de la super classe
            sélectionnerSuperClasses numéro
        End If
    Next i

End Sub
```

```
'Renvoie les contraintes des classes d'attributs composants, contenus, dispositifs,
'contiguïtés et enclos du type de la classe de parts n° "num"
'num : numéro de la classe de parts dont les classes d'attributs font l'objet de contraintes
'premier : booléen indiquant si la contrainte est la première
'opérateur : variable contenant le type d'opérateur de la contrainte

Private Function imprimerContrainteEnumération(num As Integer, premier As Boolean, _
    opérateur As String)

    'Compteur de classes d'attributs
    Dim i As Integer
    'Variable contenant les contraintes relatives aux classes d'attributs
    Dim contrainte As String
    'Booléen indiquant si la contrainte est la première
    Dim prem As Boolean

    'Initialise les variables
    contrainte = ""
    prem = premier
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs dispositifs, composants, contenus, _
        contiguïtés ou enclos
        If (tabParts(num, i, 0) Like "possible_*_features") Or _
            (tabParts(num, i, 0) Like "possible_components") Or _
            (tabParts(num, i, 0) Like "possible_contents") Or _
            (tabParts(num, i, 0) Like "contain") Or _
            (tabParts(num, i, 0) Like "possible_geometry") Or _
            (tabParts(num, i, 0) Like "contiguity") Or _
            (tabParts(num, i, 0) Like "enclosure") Then
            'Teste s'il s'agit de la première contrainte
            If prem Then
                contrainte = contrainte & " <=> " & tabParts(num, i, 1) & opérateur & "\undef"
                prem = False
            Else
                contrainte = contrainte & " \and " & tabParts(num, i, 1) & opérateur & "\undef"
            End If
        End If
    Next i
```

```
    imprimerContrainteEnumération = contrainte

End Function
```

```
'Renvoie la valeur "VRAI" si l'argument "typ" est un type prédéfini, sinon renvoie "FAUX"
'typ : nom d'une classe d'identités, de positions ou de dispositifs de valeur

Private Function typePrédéfini(typ As String)

    Select Case typ
    Case "STRING", "CHAR", "BOOLEAN", "INTEGER", "RATIONAL", "DURATION"
        typePrédéfini = True
    Case Else
        typePrédéfini = False
    End Select

End Function
```

F.8. Module standard "Module5"

'Module nommé "Module5". _
Contient deux procédure associées au processus de vérification des contraintes _
sur les relations sous-ensembles, composants, contenus et contenants. _
Elles sont respectivement appelées par le "Module1" et le "Module3".

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
'Permet de comparer des chaînes de caractères sans prendre en compte la casse
Option Compare Text

'Vérifie pour une classe de parts l'absence de cycle dans ses relations "typeRelation".
'L'algorithme vérifie particulièrement :
'la hiérarchisation des sous-classes de parts;
'0) --> circuit uniforme (relations sous-ensembles)
'que la classe de parts au départ de la procédure n'est pas composée/contenue par elle-même :
'1) soit directement;
'--> circuit uniforme (relations composants ou contenus)
'2) soit indirectement, soit qu'elle soit le composant d'une de ses super classes de parts;
'--> circuit mixte (relations sous-ensembles et composants ou contenus)
'3) soit qu'elle possède comme composant l'une de ses super classes de parts;
'--> cycle mixte (relations sous-ensembles et composants ou contenus)
'l'absence de relations multiples entre la classe de départ et une autre via l'héritage :
'4) soit qu'elle et une de ses super classes soient directement associées à une même classe;
'--> cycle mixte (relations sous-ensembles et composants ou contenus ou dispositifs physiques)
'5) soit qu'elle soit directement associée à une super classe et à une sous-classe de celle-ci;
'--> cycle mixte (relations sous-ensembles et composants, contenus, dispositifs
'géométriques, dispositifs de contiguïté ou dispositifs d'enclos)
'6) soit qu'elle et une de ses super classes soient directement associées à une super classe
'et à une sous-classe de celle-ci;
'--> cycle mixte (relations sous-ensembles et composants ou contenus)
'Num : numéro de la classe de parts à vérifier.
'pos : position du numéro de la classe de parts à l'intérieur du tableau statique "tabRel".
'nbrClasses : nombre de classes présentes dans le tableau "tabRel".
'typeClasse : type de la classe qui fait l'objet d'une vérification :
'"classe de départ" : classe à l'origine de la vérification;
'"super classe" : super classe de la classe de départ;
'"typeRelation super classe" : classe associée à "super classe" via typeRelation(s);
'"classe typeRelation" : classe associée à la "classe de départ" via typeRelation(s);
'"super classe typeRelation" : super classe d'une "classe typeRelation"
'typeRelation : type de la relation qui fait l'objet d'une vérification :
'composant, contenu et dispositif physique (relations multiples uniquement).

Sub vérifierHiérarchie(num **As Integer**, pos **As Integer**, _
nbrClasses **As Integer**, typeClasse **As String**, typeRelation **As String**)

```
'Tableau qui contient les numéros des classes de parts liées à la classe de départ _
par les relations composants, contenus ou sous-ensembles. _
(x,0) : contient le numéro d'une classe de parts. _
(x,1) : contient le booléen indiquant si la classe de parts a fait l'objet d'une vérification. _
(x,2) : string indiquant le type de la classe. _
(x,3) : entier indiquant le nombre de typeRelation entre cette classe et la classe de départ. _
"Static" car le tableau doit être conservé au travers des appels récursifs.
Static tabRel(1 To MAXTAB, 0 To 3)
'Compteur du nombre de classes d'attributs associées à une classe de parts
Dim i As Integer
'Compteur du nombre de classes présentes dans le tableau
Dim x As Integer
'Booléen indiquant s'il existe dans le tableau une classe qui n'a pas encore été vérifiée
Dim existenceClasse As Boolean
'Nombre de classes présentes dans le tableau "tabRel"
Dim nombreClasses As Integer
'Position dans le tableau "tabRel" du numéro de la classe de parts à vérifier
Dim position As Integer
'Numéro d'une classe de parts à vérifier
Dim numéro As Integer
'Booléen indiquant si une classe est déjà enregistrée dans le tableau "tabRel"
Dim déjàEnregistré As Boolean
'Variable contenant le message adressé à l'utilisateur
Dim message As String
'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
Dim typeErreur As String
'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
Dim z As Integer

'Initialise les variables
nombreClasses = nbrClasses
position = pos
'Enregistre le numéro de la classe dans le tableau
tabRel(position, 0) = num
tabRel(position, 1) = False
tabRel(position, 2) = typeClasse
'Cherche les relations "typeRelation" parmi les classes d'attributs
For i = 1 To tabParts(num, 0, 0)
    'Attention : la classe sous-ensemble hérite des composants/contenus de sa super classe, _
    laquelle doit donc aussi faire l'objet d'une recherche (via la relation "isAOrigine")
    If (tabParts(num, i, 0) Like typeRelation) Or _
        (tabParts(num, i, 0) Like "isAOrigine") Then
        'Initialise le booléen indiquant si une classe est déjà enregistrée
        déjàEnregistré = False
        'Pour toutes les classes enregistrées dans le tableau "tabRel"
        For x = 1 To nombreClasses
            'Teste si la classe de destination est déjà dans le tableau
            If (tabParts(num, i, 4) Like tabRel(x, 0)) Then
                'Renvoie le type de classe origine, le type de relation et le type de classe destination
                Select Case typeClasse & ":" & tabParts(num, i, 0) & ":" & tabRel(x, 2)
```

```
'Erreur 0 : circuit uniforme
Case "super classe:isAOrigine:classe de départ", _
    "super classe:isAOrigine:super classe"
    'Affiche le message à l'utilisateur
    message = "La hiérarchisation des classes de parts est rendue impossible " _
        & "par la présence d'un circuit formé par des relations """ _
        & "sous-ensembles"" et dont la classe de " _
        & Switch(tabParts(num, 0, 1) Like "CompoundPartClass", "parts", _
        tabParts(num, 0, 1) Like "PileOfParts", "piles", tabParts(num, 0, 1) _
        Like "Buffer", "tampons") & " """ _
        & tabParts(num, 0, 2) & """ fait partie." & Chr(10) & Chr(10) & _
        "Veuillez corriger l'erreur avant de recommencer la traduction." _
        & Chr(10) & Chr(10)
    typeErreur = "Erreur : hiérarchisation des classes de parts impossible"
    z = MsgBox(message, 0, typeErreur)
    'Force l'arrêt de la macro "traduireEnAlbertII"
    vérification = False
    Exit Sub
'Erreur 1 : circuit uniforme
Case "classe possible_components:possible_components:classe de départ", _
    "classe possible_contents:possible_contents:classe de départ"
    'Affiche le message à l'utilisateur
    message = "La classe de " & Switch(tabParts(num, 0, 1) Like _
        "CompoundPartClass", "parts", tabParts(num, 0, 1) Like "PileOfParts", _
        "piles", tabParts(num, 0, 1) Like "Buffer", "tampons") & " """ _
        & tabParts(num, 0, 2) & """ " & Switch(typeRelation Like _
        "possible_components", "est composée d'", typeRelation Like _
        "possible_contents", "se contient ") & "elle-même." & Chr(10) _
        & "Un circuit uniforme existe au sein de la relation """ _
        & typeRelation & """." & Chr(10) & Chr(10) & _
        "Veuillez corriger l'erreur avant de recommencer la traduction." _
        & Chr(10) & Chr(10)
    typeErreur = "Erreur : circuit dans la relation " & typeRelation
    z = MsgBox(message, 0, typeErreur)
    'Force l'arrêt de la macro "traduireEnAlbertII"
    vérification = False
    Exit Sub
'Erreur 2 : circuit mixte
Case "super classe:possible_components:classe de départ", _
    "super classe:possible_contents:classe de départ", _
    "possible_components super classe:isAOrigine:classe de départ", _
    "possible_contents super classe:isAOrigine:classe de départ", _
    "possible_components super classe:possible_components:classe de départ", _
    "possible_contents super classe:possible_contents:classe de départ", _
    "classe possible_components:isAOrigine:classe de départ", _
    "classe possible_contents:isAOrigine:classe de départ", _
    "super classe possible_components:isAOrigine:classe de départ", _
    "super classe possible_contents:isAOrigine:classe de départ", _
    "super classe possible_components:possible_components:classe de départ", _
    "super classe possible_contents:possible_contents:classe de départ"
```

```
        'Affiche le message à l'utilisateur
        message = "La classe de " & Switch(tabParts(tabRel(1, 0), 0, 1) Like _
            "CompoundPartClass", "parts", tabParts(tabRel(1, 0), 0, 1) Like _
            "PileOfParts", "piles", tabParts(tabRel(1, 0), 0, 1) Like "Buffer", _
            "tampons") & " """ & tabParts(tabRel(1, 0), 0, 2) & """ " & _
            Switch(typeRelation Like "possible_components", "est composée d'", _
            typeRelation Like "possible_contents", "se contient ") & "elle-même." _
            & Chr(10) & "Un circuit mixte existe au sein des relation " _
            & "sous-ensembles et """ & typeRelation & """." & Chr(10) & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : circuit mixte dans la relation " & typeRelation
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    'Erreur 3 : cycle mixte
    Case "super classe:isAOrigine:classe possible_components", _
        "super classe:isAOrigine:classe possible_contents", _
        "classe possible_components:possible_components:super classe", _
        "classe possible_contents:possible_contents:super classe", _
        "super classe possible_components:possible_components:super classe", _
        "super classe possible_contents:possible_contents:super classe"
        'Affiche le message à l'utilisateur
        message = "La classe de " & Switch(tabParts(tabRel(1, 0), 0, 1) Like _
            "CompoundPartClass", "parts", tabParts(tabRel(1, 0), 0, 1) Like _
            "PileOfParts", "piles", tabParts(tabRel(1, 0), 0, 1) Like "Buffer", _
            "tampons") & " """ & tabParts(tabRel(1, 0), 0, 2) & """ " _
            & Switch(typeRelation Like "possible_components", "est composée d'", _
            typeRelation Like "possible_contents", "se contient ") & "elle-même." _
            & Chr(10) & "Un cycle mixte existe au sein des relation " _
            & "sous-ensembles et """ & typeRelation & """." & Chr(10) & Chr(10) & _
            "Veuillez corriger l'erreur avant de recommencer la traduction." _
            & Chr(10) & Chr(10)
        typeErreur = "Erreur : cycle mixte dans la relation " & typeRelation
        z = MsgBox(message, 0, typeErreur)
        'Force l'arrêt de la macro "traduireEnAlbertII"
        vérification = False
        Exit Sub
    'Erreur 4
    Case "super classe:possible_components:classe possible_components", _
        "super classe:possible_contents:classe possible_contents", _
        "super classe:possible_physical_features:classe possible_physical_features"
        'Toujours 1 typeRelation entre les classes de destination et de départ _
        car l'algorithme vérifie d'abord les super classes de la classe de départ
        'Affiche le message à l'utilisateur
        message = "Plusieurs relations existent entre la classe de " _
            & Switch(tabParts(tabRel(1, 0), 0, 1) Like "BasicOrCompoundPartClass", _
            "parts mixtes", tabParts(tabRel(1, 0), 0, 1) Like "BasicPartClass", _
            "parts de base", tabParts(tabRel(1, 0), 0, 1) Like "CompoundPartClass", _
```

```
                "parts composées", tabParts(tabRel(1, 0), 0, 1) Like "PileOfParts", _
                "piles", tabParts(tabRel(1, 0), 0, 1) Like "Container", "conteneurs", _
                tabParts(tabRel(1, 0), 0, 1) Like "Buffer", "tampons") & " """ _
                & tabParts(tabRel(1, 0), 0, 2) & """ et la classe """ _
                & tabParts(tabRel(x, 0), 0, 2) & """." & Chr(10) _
                & "La classe de " & Switch(tabParts(tabRel(1, 0), 0, 1) Like _
                "PileOfParts", "piles", tabParts(tabRel(1, 0), 0, 1) Like "Container", _
                "conteneurs", tabParts(tabRel(1, 0), 0, 1) Like "Buffer", "tampons", _
                True, "parts") & " hérite ces relations de ses super classes." _
                & Chr(10) & Chr(10) & _
                "Veuillez corriger l'erreur avant de recommencer la traduction." _
                & Chr(10) & Chr(10)
            typeErreur = "Erreur : relations multiples"
            z = MsgBox(message, 0, typeErreur)
            'Force l'arrêt de la macro "traduireEnAlbertII"
            vérification = False
            Exit Sub
        'Erreur 5
        Case "classe possible_components:isAOrigine:classe possible_components", _
            "classe possible_contents:isAOrigine:classe possible_contents", _
            "super classe possible_components:isAOrigine:classe possible_components", _
            "super classe possible_contents:isAOrigine:classe possible_contents", _
        "classe possible_geometrical_features:isAOrigine:classe possible_geometrical_features", _
            "classe contiguity_feature:isAOrigine:classe contiguity_feature", _
            "classe enclosure_feature:isAOrigine:classe enclosure_feature", _
        "super classe possible_geometrical_features:isAOrigine:classe possible_geometrical_features", _
            "super classe contiguity_feature:isAOrigine:classe contiguity_feature", _
            "super classe enclosure_feature:isAOrigine:classe enclosure_feature"
            'Teste si 1 typeRelation entre les classes de destination et de départ _
            et entre les classes d'origine et de départ
            If (tabRel(x, 3) = 1) And (tabRel(position, 3) = 1) Then
                'Affiche le message à l'utilisateur
                message = "Plusieurs relations existent entre la classe de " _
                    & Switch(tabParts(tabRel(1, 0), 0, 1) Like "CompoundPartClass", _
                    "parts", tabParts(tabRel(1, 0), 0, 1) Like "PileOfParts", "piles", _
                    tabParts(tabRel(1, 0), 0, 1) Like "Container", "conteneurs", _
                    tabParts(tabRel(1, 0), 0, 1) Like "Buffer", "tampons", _
                    tabParts(tabRel(1, 0), 0, 1) Like "PartGeometricalFeatureClass", _
                    "dispositifs géométriques", _
                    tabParts(tabRel(1, 0), 0, 1) Like "PartContiguityFeature", _
                    "dispositifs de contiguïté", tabParts(tabRel(1, 0), 0, 1) _
                    Like "PartEnclosureFeature", "dispositifs d'enclos") & " """ _
                    & tabParts(tabRel(1, 0), 0, 2) & """ et la classe (ou une " _
                    & "sous-classe de) """ & tabParts(num, 0, 2) & """." & Chr(10) _
                    & "Cette dernière hérite ces relations de ses super classes." _
                    & Chr(10) & Chr(10) & _
                    "Veuillez corriger l'erreur avant de recommencer la traduction." _
                    & Chr(10) & Chr(10)
                typeErreur = "Erreur : relations multiples"
                z = MsgBox(message, 0, typeErreur)
```

```
                    'Force l'arrêt de la macro "traduireEnAlbertII"
                    vérification = False
                    Exit Sub
                Else
                    déjàEnregistré = True
                End If
            'Erreur 6
            Case "possible_components super classe:isAOrigine:classe possible_components", _
                "possible_contents super classe:isAOrigine:classe possible_contents", _
                "classe possible_components:isAOrigine:possible_components super classe", _
                "classe possible_contents:isAOrigine:possible_contents super classe", _
                "super classe possible_components:isAOrigine:possible_components super classe", _
                "super classe possible_contents:isAOrigine:possible_contents super classe"
                'Teste si 1 typeRelation entre les classes de destination et de départ _
                et entre les classes d'origine et de départ
                If (tabRel(x, 3) = 1) And (tabRel(position, 3) = 1) Then
                    'Affiche le message à l'utilisateur
                    message = "Plusieurs relations existent entre la classe de " _
                        & Switch(tabParts(tabRel(1, 0), 0, 1) Like "CompoundPartClass", _
                        "parts", tabParts(tabRel(1, 0), 0, 1) Like "PileOfParts", "piles", _
                        tabParts(tabRel(1, 0), 0, 1) Like "Container", "conteneurs", _
                        tabParts(tabRel(1, 0), 0, 1) Like "Buffer", "tampons") & " """ _
                        & tabParts(tabRel(1, 0), 0, 2) & """ et la classe """ _
                        & tabParts(num, 0, 2) & """." & Chr(10) _
                        & "Les classes héritent ces relations de leurs super classes." _
                        & Chr(10) & Chr(10) & _
                        "Veuillez corriger l'erreur avant de recommencer la traduction." _
                        & Chr(10) & Chr(10)
                    typeErreur = "Erreur : relations multiples"
                    z = MsgBox(message, 0, typeErreur)
                    'Force l'arrêt de la macro "traduireEnAlbertII"
                    vérification = False
                    Exit Sub
                Else
                    déjàEnregistré = True
                End If
            Case Else
                déjàEnregistré = True
            End Select
        End If
    Next x
    'Teste si la relation ne se trouve pas encore dans le tableau
    If Not (déjàEnregistré) Then
        '1) Met à jour le compteur de classes
        nombreClasses = nombreClasses + 1
        '2) Enregistre la classe dans le tableau
        tabRel(nombreClasses, 0) = tabParts(num, i, 4)
        tabRel(nombreClasses, 1) = False
        Select Case tabRel(position, 2)
        Case "classe de départ"
```

```
        'Teste s'il s'agit de la relation sous-ensemble
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            tabRel(nombreClasses, 2) = "super classe"
            tabRel(nombreClasses, 3) = 0
        Else
            tabRel(nombreClasses, 2) = "classe " & typeRelation
            tabRel(nombreClasses, 3) = 1
        End If
    Case "super classe"
        'Teste s'il s'agit de la relation sous-ensemble
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            tabRel(nombreClasses, 2) = "super classe"
            tabRel(nombreClasses, 3) = 0
        Else
            tabRel(nombreClasses, 2) = typeRelation & " super classe"
            tabRel(nombreClasses, 3) = 1
        End If
    Case "possible_components super classe", "possible_contents super classe", _
        "possible_physical_features super classe"
        tabRel(nombreClasses, 2) = typeRelation & " super classe"
        'Teste s'il s'agit de la relation sous-ensemble
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            tabRel(nombreClasses, 3) = tabRel(position, 3)
        Else
            tabRel(nombreClasses, 3) = tabRel(position, 3) + 1
        End If
    Case "classe possible_components", "classe possible_contents", _
        "classe possible_physical_features", "classe possible_geometrical_features", _
        "classe contiguity_feature", "classe enclosure_feature"
        'Teste s'il s'agit de la relation sous-ensemble
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            tabRel(nombreClasses, 2) = "super classe " & typeRelation
            tabRel(nombreClasses, 3) = tabRel(position, 3)
        Else
            tabRel(nombreClasses, 2) = "classe " & typeRelation
            tabRel(nombreClasses, 3) = tabRel(position, 3) + 1
        End If
    Case "super classe possible_components", "super classe possible_contents", _
        "super classe possible_geometrical_features", "super classe contiguity_feature", _
        "super classe enclosure_feature"
        tabRel(nombreClasses, 2) = "super classe " & typeRelation
        'Teste s'il s'agit de la relation sous-ensemble
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            tabRel(nombreClasses, 3) = tabRel(position, 3)
        Else
            tabRel(nombreClasses, 3) = tabRel(position, 3) + 1
        End If
    Case Else
        Debug.Print "Le type de classes n'est pas prévu."
    End Select
```

```
            End If
        End If
    Next i

    'Met à jour le booléen indiquant que la classe de parts a été vérifiée
    tabRel(position, 1) = True

    'Initialise les variables
    existenceClasse = False
    i = 1
    'Cherche dans le tableau une super classe de la classe de départ qui n'a pas encore été vérifiée
    While Not (existenceClasse) And (i <= nombreClasses)
        If Not (tabRel(i, 1)) And (tabRel(i, 2) Like "super classe") Then
            'Lance la vérification
            numéro = tabRel(i, 0)
            message = tabRel(i, 2)
            vérifierHiérarchie numéro, i, nombreClasses, message, typeRelation
            existenceClasse = True
        End If
        i = i + 1
    Wend
    'Initialise le compteur de classes
    i = 1
    'Cherche dans le tableau une classe qui n'a pas encore été vérifiée
    While Not (existenceClasse) And (i <= nombreClasses)
        If Not (tabRel(i, 1)) Then
            'Lance la vérification
            numéro = tabRel(i, 0)
            message = tabRel(i, 2)
            vérifierHiérarchie numéro, i, nombreClasses, message, typeRelation
            existenceClasse = True
        End If
        i = i + 1
    Wend

End Sub
```

```
'Vérifie qu'une part composée ou une pile peut être constituée de plusieurs composants.
'Donc la somme des bornes supérieures des cardinalités des relations composants d'une classe de
'parts composées/piles et de ses super classes éventuelles doit être supérieure ou égale à deux.
'Vérifie aussi qu'un conteneur peut contenir plusieurs contenus via les relations contenants.
'num : numéro de la classe de parts composées, de piles ou de conteneurs à vérifier
'relation : type de relation (composant ou contenant)

Public Sub vérifierComposants(num As Integer, relation As String)

    'Booléen indiquant si la contrainte est vérifiée _
    "Static" car la valeur doit être conservée au travers des appels récursifs
    Static contrainte As Boolean
```

```
    'Variable contenant la somme des bornes supérieures des cardinalités des relations _
    "Static" car la valeur doit être conservée au travers des appels récursifs
    Static somme As Integer
    'Compteur du nombre de relations associées à une classe de parts
    Dim i As Integer
    'Numéro d'une classe de parts composées, de piles ou de conteneurs à vérifier
    Dim numéro As Integer

    'Intialise les variables
    somme = 0
    contrainte = False
    'Pour toutes les classes d'attributs de la classe de parts composées, de piles ou de conteneurs
    For i = 1 To tabParts(num, 0, 0)
        'Teste s'il s'agit d'une classe d'attributs composants ou contenants
        If (tabParts(num, i, 0) Like relation) Then
            'Teste si la borne supérieure est un nombre indéfini
            If (tabParts(num, i, 3) Like "N") Then
                contrainte = True
            Else
                'Sinon comptabilise les bornes
                somme = somme + tabParts(num, i, 3)
            End If
        End If
        'Teste si la classe de parts composées, de piles ou de conteneurs est un sous-ensemble
        If (tabParts(num, i, 0) Like "isAOrigine") Then
            'Poursuit la vérification au niveau de la super classe _
            L'algorithme fonctionne correctement car il remonte jusqu'à la super classe racine _
            puis il effectue les calculs en redescendant.  Il remonte d'abord car la relation _
            "isAOrigine" est la première dans le tableau.  Donc les deux variables statiques _
            ont été initialisées (mises à zéro et à faux) avant les calculs.
            numéro = tabParts(num, i, 4)
            vérifierComposants numéro, relation
        End If
    Next i
    'Teste si la somme des bornes supérieures est supérieure ou égale à deux
    If (somme >= 2) Then
        contrainte = True
    End If
    'Enregistre le résultat de la vérification
    vérification = contrainte

End Sub
```

F.9. Module standard "Module6"

```
'Cette macro nommée "Module6" ajoute trois boutons à la barre d'outils du dessin : _
1 : exécute la macro de traduction du modèle graphique de "parts" en langage AlbertII _
2 : exécute la macro ajoutant une page au dessin _
3 : exécute la macro supprimant la page active (vide) du dessin.

'Impose la déclaration explicite de toutes les variables du module
Option Explicit
```

```
Public Sub ajouterBoutons()

    'Objet d'interface utilisateur utilisé pour la copie des barres d'outils et d'états MSOffice
    Dim barres As Visio.UIObject
    'Jeu de barres d'outils (et pas d'états) de la fenêtre active
    Dim jeuBarres As Visio.ToolbarSet
    'Collection d'éléments figurant sur une barre d'outils
    Dim collectionBoutons As Visio.ToolbarItems
    'Nouveau bouton à ajouter sur la barre d'outils pour déclencher la macro de traduction en AlbertII
    Dim bouton As Visio.ToolbarItem

    'Renvoie un objet "UIObject" qui représente une copie des barres d'outils intégrées de Visio
    Set barres = Visio.Application.BuiltInToolbars(visToolBarMSOffice)

    'Renvoie le jeu de barres d'outils de la fenêtre de dessin
    'Comprend un objet Toolbar pour chaque barre d'outils figurant dans un contexte de fenêtre
    Set jeuBarres = barres.ToolbarSets.ItemAtID(visUIObjSetDrawing)

    'PREMIER BOUTON : traduction du modèle graphique de "parts" en langage AlbertII
    'Renvoie la collection d'éléments figurant sur la première barre d'outils (barre n° 0)
    Set collectionBoutons = jeuBarres.Toolbars(0).ToolbarItems

    'Ajoute un bouton en dernière position (contrairement aux habitudes, l'index débute à 0 et non à 1)
    Set bouton = collectionBoutons.AddAt(collectionBoutons.Count)

    'Définit les propriétés du nouveau bouton de barre d'outils
    'Affiche ou définit le type de contrôle d'un élément de barre d'outils (ici, un bouton)
    bouton.CntrlType = visCtrlTypeBUTTON
    'Affiche ou définit le texte du bouton
    bouton.ActionText = "Exécute la traduction du modèle graphique de ""parts"" en langage AlbertII"
    'Détermine que le bouton affiche une légende
    bouton.Style = 2
    'Affiche ou définit la légende du bouton
    bouton.Caption = "Traduction en AlbertII"
    'Affiche ou définit le nom de la macro associée au bouton ("Module1")
    bouton.AddOnName = "traduireEnAlbertII"
```

```
    'DEUXIEME BOUTON : ajout d'une page au modèle graphique de "parts"
    'Renvoie la collection d'éléments figurant sur la deuxième barre d'outils (barre n° 1)
    Set collectionBoutons = jeuBarres.Toolbars(1).ToolbarItems

    'Ajoute un bouton en dernière position (contrairement aux habitudes, l'index débute à 0 et non à 1)
    Set bouton = collectionBoutons.AddAt(collectionBoutons.Count)

    'Définit les propriétés du nouveau bouton de barre d'outils
    'Affiche ou définit le type de contrôle d'un élément de barre d'outils (ici, un bouton)
    bouton.CntrlType = visCtrlTypeBUTTON
    'Affiche ou définit le texte du bouton
    bouton.ActionText = "Ajoute une page au modèle graphique de ""parts"""
    'Détermine que le bouton affiche une légende
    bouton.Style = 2
    'Affiche ou définit la légende du bouton
    bouton.Caption = "Ajout page"
    'Affiche ou définit le nom de la macro associée au bouton ("Module7")
    bouton.AddOnName = "ajouterPage"

    'TROISIEME BOUTON : suppression de la page active (vide) du modèle graphique de "parts"
    'Renvoie la collection d'éléments figurant sur la deuxième barre d'outils (barre n° 1)
    Set collectionBoutons = jeuBarres.Toolbars(1).ToolbarItems

    'Ajoute un bouton en dernière position (contrairement aux habitudes, l'index débute à 0 et non à 1)
    Set bouton = collectionBoutons.AddAt(collectionBoutons.Count)

    'Définit les propriétés du nouveau bouton de barre d'outils
    'Affiche ou définit le type de contrôle d'un élément de barre d'outils (ici, un bouton)
    bouton.CntrlType = visCtrlTypeBUTTON
    'Affiche ou définit le texte du bouton
    bouton.ActionText = "Supprime la page active (vide) du modèle graphique de ""parts"""
    'Détermine que le bouton affiche une légende
    bouton.Style = 2
    'Affiche ou définit la légende du bouton
    bouton.Caption = "Suppression page"
    'Affiche ou définit le nom de la macro associée au bouton ("Module7")
    bouton.AddOnName = "supprimerPage"

    'Demande à Visio d'utiliser la nouvelle interface utilisateur personnalisée
    ThisDocument.SetCustomToolbars barres

End Sub
```

F.10. Module standard "Module7"

'Cette macro nommée "Module7" contient deux procédures : la première ajoute une page _
au dessin tandis que la seconde supprime une page (vide) du dessin.

'Impose la déclaration explicite de toutes les variables du module
Option Explicit

'Ajoute une page au dessin (depuis un bouton sur la barre d'outils défini au "Module6").
'La page est placée après la page active tandis que les pages suivantes sont renumérotées.

```
Sub ajouterPage()

    'Variable contenant la collection de pages du document actif (modèle graphique ou dessin)
    Dim collectionPages As Visio.Pages
    'Variable contenant la nouvelle page à ajouter au document actif
    Dim nouvellePage As Visio.page
    'Variable contant le nombre de pages du document actif
    Dim nombrePages As Integer
    'Variable contenant le numéro de la page active
    Dim num As Integer
    'Compteur de pages
    Dim i As Integer

    'Renvoie le numéro de la page active
    num = Visio.ActivePage.Index
    'Renvoie la collection de pages du document actif
    Set collectionPages = Visio.ActiveDocument.Pages
    'Ajoute une page à la collection de pages du document actif
    Set nouvellePage = collectionPages.Add

    'Renvoie le nombre de pages du document
    nombrePages = Visio.ActiveDocument.Pages.Count
    'Place la nouvelle page juste après la page active
    Visio.ActiveDocument.Pages(nombrePages).Index = num + 1
    'Renumérote les pages qui suivent la page active
    For i = nombrePages To num + 1 Step -1
        Visio.ActiveDocument.Pages(i).Name = i
    Next i
    'Définit la taille d'affichage de la page (75 %)
    Visio.ActiveWindow.Zoom = 0.75

End Sub
```

```
'Supprime une page du dessin (depuis un bouton sur la barre d'outils défini au "Module6").
'Préalable 1 : la page supprimée est la page active et elle doit être vide (sans forme).
'Préalable 2 : si le projet ne contient plus qu'une seule page, elle ne peut être supprimée.

Sub supprimerPage()

    'Variable contant le nombre de pages du document actif
    Dim nombrePages As Integer
    'Variable contenant le numéro de la page supprimée
    Dim num As Integer
    'Compteur de pages
    Dim i As Integer
    'Variable contenant le message adressé à l'utilisateur
    Dim message As String
    'Variable contenant le type d'erreur placé dans le titre du message de l'utilisateur
    Dim typeErreur As String
    'Variable indispensable à la fonction "MsgBox" mais inutile dans ce code
    Dim z As Integer

    'Teste si la page active est vide
    If (Visio.ActivePage.Shapes.Count = 0) Then
        'Teste si la page n'est pas la seule page du projet
        If Not (Visio.ActiveDocument.Pages.Count < 2) Then
            'Renvoie le nombre de pages du document
            nombrePages = Visio.ActiveDocument.Pages.Count
            'Renvoie le numéro de la page à supprimer
            num = Visio.ActivePage.Index
            'Supprime la page active
            Visio.ActivePage.Delete (0)
            'Renumérote les pages suivantes
            For i = num To nombrePages - 1
                Visio.ActiveDocument.Pages(i).Name = i
            Next i
        Else
            'Affiche un message à l'utilisateur si le projet ne contient plus qu'une seule page
            message = "Vous ne pouvez pas supprimer la seule page du modèle." & Chr(10)
            typeErreur = "Erreur : page unique"
            z = MsgBox(message, 0, typeErreur)
        End If
    Else
        'Affiche un message à l'utilisateur si la page n'est pas vide
        message = "Pour pouvoir être supprimée, une page doit être vide." & Chr(10) & _
            "Or la page active contient encore " & Visio.ActivePage.Shapes.Count _
            & " formes." & Chr(10) & Chr(10)
        typeErreur = "Erreur : page non vide"
        z = MsgBox(message, 0, typeErreur)
    End If

End Sub
```